The Z8 Application Note Handbook

Retro Workshop
果粉工作室

THE Z8 APPLICATION NOTE HANDBOOK

果粉工作室
Retro Workshop

© 1996 by Zilog, Inc. All rights reserved. No part of this document may be copied or reproduced in any form or by any means without the prior written consent of Zilog, Inc. The information in this document is subject to change without notice. Devices sold by Zilog, Inc. are covered by warranty and patent indemnification provisions appearing in Zilog, Inc. Terms and Conditions of Sale only. Zilog, Inc. makes no warranty, express, statutory, implied or by description, regarding the information set forth herein or regarding the freedom of the described devices from intellectual property infringement. Zilog, Inc. makes no warranty of merchantability or fitness for any purpose. Zilog, Inc. shall not be responsible for any errors that may appear in this document. Zilog, Inc. makes no commitment to update or keep current the information contained in this document.

Zilog's products are not authorized for use as critical components in life support devices or systems unless a specific written agreement pertaining to such intended use is executed between the customer and Zilog prior to use. Life support devices or systems are those which are intended for surgical implantation into the body, or which sustains life whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in significant injury to the user.

Zilog, Inc. 210 East Hacienda Ave.
Campbell, CA 95008-6600
Telephone (408) 370-8000
FAX 408 370-8056
Internet: http://www.zilog.com

Retro Workshop
果粉工作室

THE Z8 APPLICATION NOTE HANDBOOK

TABLE OF CONTENTS

TITLE	SECTION
INTRODUCTION	ii
USING THE ZILOG Z8 CCP MCU AS AN I^2C BUS MASTER	1-1
USING SOFTWARE DESIGN TECHNIQUES TO MAXIMIZE Z8 SYSTEM NOISE IMMUNITY	2-1
PIC16C54-TO-Z86E02 UPGRADE	3-1
USING A Z8 MCU FOR POWER PHASE CONTROL	4-1
ANALOG-TO-DIGITAL CONVERSION TECHNIQUES USING ZILOG Z8 MCUS	5-1
A DC MOTOR CONTROLLER USING THE ZILOG Z86E06 MCU	6-1
ZILOG Z8 I/O EXPANDER MODULE	7-1
DIGITAL INSTANT WATER HEATER	8-1
ON-CHIP OSCILLATOR DESIGN	9-1
THE Z86C08 CONTROLS A SCROLLING LED MESSAGE DISPLAY	10-1
DTMF TONE GENERATION USING THE Z8 CCP	11-1
TIMEKEEPING WITH THE Z8	12-1
SERIAL COMMUNICATIONS USING THE Z8 CCP SOFTWARE UART	13-1
INTERFACING LCDS TO THE Z8	14-1
THE VERSATILE Z86C08: THREE KEY FEATURES OF THIS Z8 MICROCONTROLLER	15-1
DESIGNING A LOW-COST THERMAL PRINTER	16-1
ZILOG SALES OFFICES, REPRESENTATIVES, AND DISTRIBUTORS	Z
LITERATURE GUIDE AND ORDERING INFORMATION	L

INTRODUCTION

ZILOG'S FOCUS ON APPLICATION-SPECIFIC PRODUCTS HELPS YOU MAINTAIN YOUR TECHNOLOGICAL EDGE

Zilog's Z8 family of products offer low cost, small die sizes, and formidable computing power to a wide range of consumer and industrial applications.

Z8 APPLICATION NOTES

Zilog's ASSPs are optimized for maximum functionality in specific applications. To develop an understanding of the technology of the application, Zilog builds actual end products. Written from our experience developing proven designs, Zilog's application notes offer a useful guide for designers.

ABOUT THIS HANDBOOK

The following individuals have contributed to the writing and technical review of the application notes appearing in this handbook:

Authors: Ivan Chiang, Daniel Mui, Don Newquist, Bill Soto, and Jon Veres.

Technical Reviewers: Edwin Aquino, Ivan Chiang, Gary Cummings, Pete Gilmour, John Goodrich, and Au Truong.

Roger Dills, Editor
Zilog Marketing Communications

USING THE ZILOG Z8 CCP MCU AS AN I²C BUS MASTER

WHEN A STANDARDIZED PERIPHERAL BUS CONNECTION IS NEEDED, THE I²C SERIAL BUS CAN BE EASILY SIMULATED BY SOFTWARE CONTROL OF TWO Z8® CCP™ MCU I/O PINS.

INTRODUCTION

The Zilog Z8® Consumer Controller Processor (CCP™) family has many integrated features that simplify system design requirements in embedded applications. These on-chip peripheral features include Power-On Reset (POR), Low-Voltage Protection, Watch-Dog Timer (WDT), programmable I/O, comparators, multiple timer/counters (including external event counting and output waveform generation), and multiple external/internal interrupts. Some applications, however, may require additional external peripheral support, such as an I/O expander, specialty memory, LCD driver, or Analog-to-Digital (A-D) converter. To provide a standardized peripheral bus connection, a simple bidirectional, two-wire serial interface called the Inter-IC (I²C) bus may be used. While the Z8 CCP MCU family members do not possess dedicated on-chip I²C support hardware, this serial bus may be easily simulated by software control of two MCU I/O pins.

This application note provides an overview of the I²C bus interface and the 24CXXX EEPROM family, and details an application interfacing the Z8 CCP MCU (I²C Master) to a two-wire 24CXXX EEPROM (I²C Slave).

I²C BUS OVERVIEW

The I²C bus uses a two-wire interface consisting of a serial data line (SDA) and a serial clock line (SCL) to exchange information between devices connected to the bus. Each device on the bus has its own unique address and can operate as a transmitter or receiver (depending on its particular function). Devices are further characterized as Masters or Slaves. A "Master" is defined as a device that initiates, controls (generates all framing and clock signals), and terminates a transfer. A "Slave" is defined as any device addressed by a Master. Multiple Masters are allowed by I²C bus specifications because a bus arbitration and clock synchronization scheme is defined so that messages are not corrupted when more than one device attempts to take Master control. This procedure is also facilitated because all connections to the bus are wired-AND connections.

Both the SDA and SCL lines are bidirectional and are pulled up to the positive logic supply rail (via pull-up resistors, both lines High for the bus idle state). When in the output mode, the SDA and SCL lines are open-drain or open-collector. Data is exchanged 8 bits at a time and can be transferred at rates up to 100 Kbps (standard mode) or 400 Kbps (fast mode).

Data transfers on the I²C bus are controlled (and framed) via two unique bus states generated by the bus Master—START and STOP bit conditions. A START condition is defined as a High- to Low-level transition on SDA while the SCL line is High. A STOP condition is defined as a Low-to High-level transition on SDA while the SCL line is High (refer to Figure 1). Data must always be valid (stable) on the SDA line during the SCL High time and that it is only allowed to be changed during the SCL Low period (refer to Figure 2). Thus, one bit of data is transmitted for each SCL period.

Retro Workshop 果粉工作室

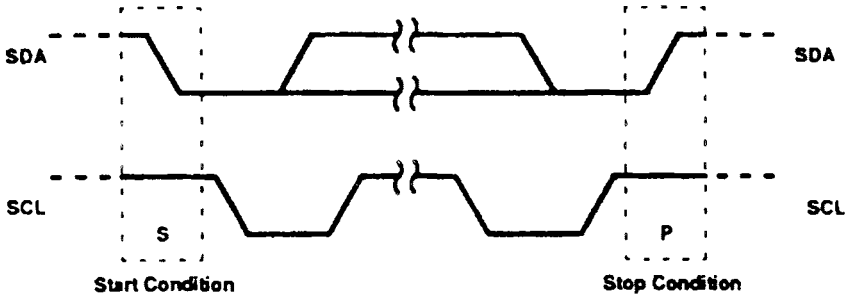


Figure 1. START and STOP Conditions

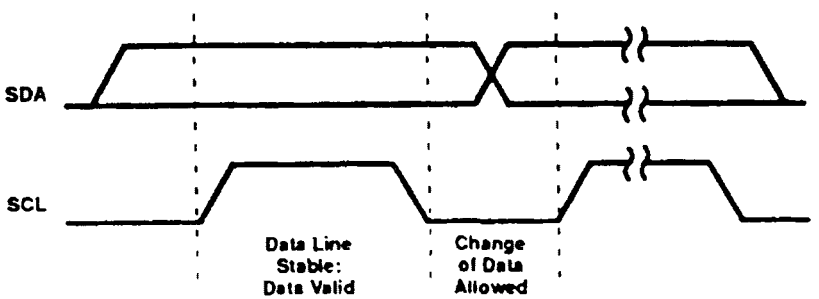


Figure 2. Valid Bit Data on the I²C Bus

Following the START condition, the first 8-bit byte sent in a bus message is a 7-bit Slave address field along with a data direction or R/W bit. (This discussion is limited to the 7-bit addressing mode.) The data direction bit (least significant bit) controls whether or not the Master will transmit (0 = write) or receive (1 = read) data from the addressed Slave. For every 8-bit byte exchanged, the most significant bit is always transmitted first. All 8-bit byte bus transactions (whether address, data, and so on) are followed by an acknowledge bit (refer to Figure 3). The acknowledge bit is a low-level signal placed on the SDA line by the receiving device (Master or Slave) during the Master transmitted acknowledge clock pulse (ninth High SCL clock pulse of the byte transmission).

If the receiver is unable to receive data (busy Slave receiver) or must signal the end-of-data condition (Master receiver), a non-acknowledge is sent (SDA High during the ninth High SCL clock pulse time, as shown in Figure 3). Following the START and Slave address transmission, data is exchanged between the Master and receiver as required. Upon exchange of the final byte and its acknowledge, the Master issues the STOP condition to end bus usage. (See Figure 4 for a sample I²C bus data transfer session.)

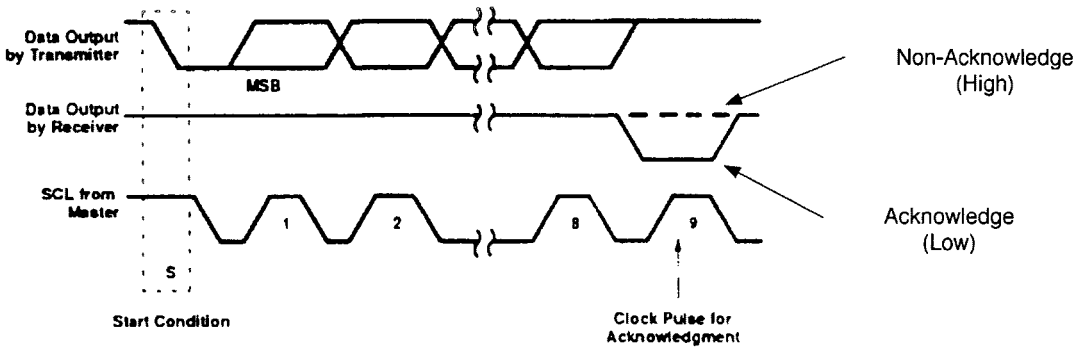


Figure 3. I²C Bus Acknowledge, Non-Acknowledge

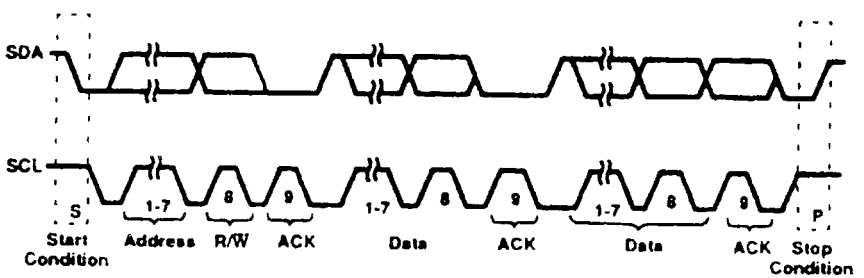


Figure 4. Sample I²C Bus Message

The previous general overview of I²C bus operation provides the foundation necessary to proceed with a serial EEPROM interface (a working knowledge of the I²C 7-bit addressing mode).

For complete details of all the various modes this bus supports, see the I²C Bus Specifications published by Philips Corporation, referenced at the conclusion of this app. note.

24CXXX SERIAL EEPROM OVERVIEW

The 24CXXX Family of two-wire serial EEPROM devices provide 256 to 2K (24C01A to 24C16) bytes of non-volatile storage and are I²C bus compatible. The standard 8-pin PDIP package pinout is shown in Figure 5.

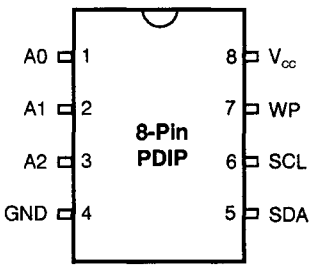


Figure 5. 24CXXX EEPROM 8-Pin PDIP Pinout

In addition to SDA and SCL, device control pins include:

- Write Protect (WP) Input. When this pin is grounded, normal read/write EEPROM accesses are permitted. When connected to V_{cc}, the EEPROM is write protected.
- A2, A1, A0: Device Address Inputs. All Ax address pins are hard-wired to fix the device's programmable software address.

EEPROM ADDRESSING

Data is exchanged with the 24CXXX as a standard I²C bus Slave device. After the START command, the first byte transmitted is the EEPROM's address byte. The upper nibble identifies the device as memory (per the I²C specification). The lower nibble selects which 24CXXX is targeted for exchange (Ax, up to 8 devices may be on the bus) and/or which internal 256-byte block in the EEPROM (Px). (See Table 1 for address byte definition.)

Table 1. 24CXXX Device Name/Address Byte Definition

Device	No. of Bytes	Max. No. of Devs.	Page Write (No. of Bytes)	Device Address Byte (MSB to LSB)
24C01A	128	8	8	1010 A2 A1 A0 R/W
24C02	256	8	8	1010 A2 A1 A0 R/W
24C04	512	4	16	1010 A2 A1 P0 R/W
24C08	1K	2	16	1010 A2 P1 P0 R/W
24C16	2K	1	16	1010 P2 P1 P0 R/W
R/W: 0 = Write; 1 = Read. Ax = Hard-Wired Device Address. Px = Internal Software Address. (Selects which 256-Byte Block to Address.)				

After the device address is transmitted, the desired 8-bit EEPROM word address is sent. At this point the desired device operation—read or write—is carried out.

Write Operations

Two types of write operations (byte and page) are supported (see Figure 6). Byte operations allow a random EEPROM address to be written. Byte write operations require transmission of the following:

- START Condition (Master)
- EEPROM Device Address with R/W Bit = 0 (Master)
- Acknowledge Bit (EEPROM)
- Target EEPROM Word Address to be Written (Master)
- Acknowledge Bit (EEPROM)
- Data Byte to be Written (Master)
- Acknowledge Bit (EEPROM)
- STOP Condition (Master)

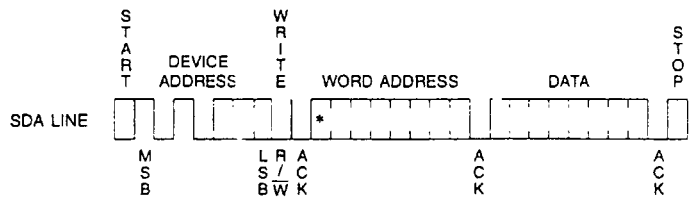


Figure 6. Byte Write Operation

Note that upon detecting the STOP bit, the EEPROM enters an internally timed write cycle to non-volatile memory. During this write cycle, all EEPROM inputs are disabled and the device will not respond to further bus activity until the write is complete. The Master device has two ways to determine when the EEPROM internal write cycle is completed: 1) timeout the maximum required write time (usually much longer than the device actually requires), 2) perform and acknowledge poll, which is the more time efficient way. This involves sending a start condition followed by the target EEPROM address. If the EEPROM acknowledges its address, the device write cycle is complete and the Master's desired operation may continue. (Otherwise, polling must be continued.)

Page write operations are identical to the byte write operations described previously, except that instead of writing one data byte, as many as 8 or 16 bytes (up to the maximum page write size the EEPROM present supports) may be continuously written after the EEPROM address and before the STOP bit transmission. (Refer to Table 1 and Figure 7.) During these page writes, the EEPROM automatically increments the address pointer between bytes.

Note: If the device's maximum page number is exceeded, the data will simply wrap around.

Once the STOP condition is received, those bytes updated will be written to the nonvolatile array. The great advantage of the page write feature is that only one write-cycle timing is consumed to write multiple bytes of data.

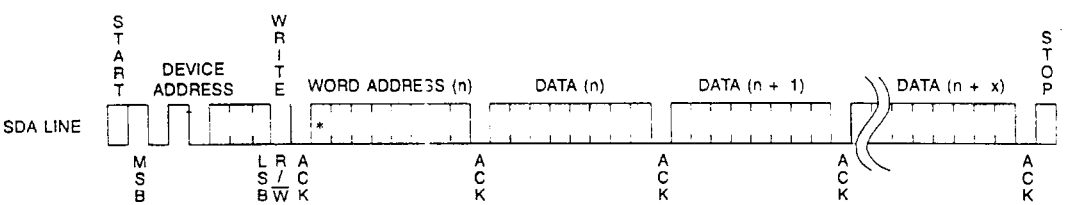


Figure 7. Page Write Operation

Read Operations

Three types of read operations are supported: Current Address, Random, and Sequential Read. Read operations begin just like write operations except the device address byte has the R/W bit set to 1.

For the Current Address Read mode, no EEPROM byte address is written as the data transmitted by the addressed Slave to the Master is read from the location of last access (incremented by one). This type read transmission sequence appears as follows:

- START Condition (Master)
- EEPROM Device Address with R/W Bit = 1 (Master)
- Acknowledge Bit (EEPROM)
- Data Byte to be Read (EEPROM: bytes sent are from the addressed Slaves last pointed to memory location incremented by 1)
- Non-Acknowledge Bit (Master)
- STOP Condition (Master)
- The Master signals a non-acknowledge and STOP condition to terminate data exchange with the Slave EEPROM (see Figure 8).

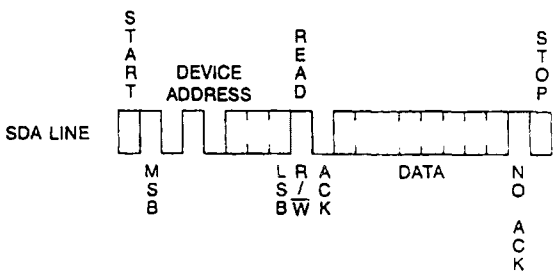
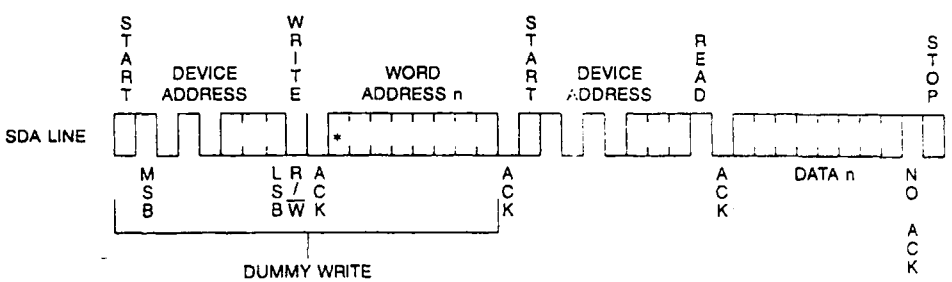


Figure 8. Current Address Read Operation

The Random Read mode is begun with a dummy byte write cycle (Master sends a START condition followed by the device address and target word address) followed by a Current Address Read mode cycle as described previously (see Figure 9).



(* = Don't care bit for 1K)

Figure 9. Random Read Operation

The Sequential Read mode is initiated with either a Current Address Read or Random Read. In this case, instead of the Master terminating the read after a single byte exchange (with a non-acknowledge), the Master responds with a valid acknowledge after each received data byte. This instructs the Slave EEPROM to continue the read operation and transmit out the next data byte. Sequential reads continue until terminated by the Master via issuance of a non-acknowledge on the last desired byte read followed by the STOP condition (see Figure 10).

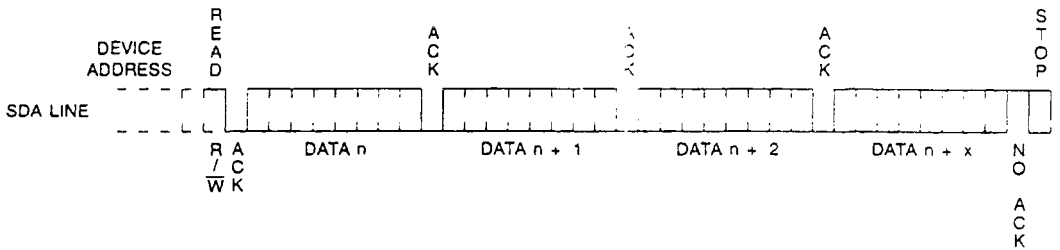


Figure 10. Sequential Read Operation

Z8 CCP MCU INTERFACE TO 24CXXX EEPROM

The hardware interface between the Z8® CCP™ MCU and a 24CXXX EEPROM is shown in Figure 11. The Write Protect (WP) input is tied to logic ground so that the MCU may write the EEPROM. Address line A0, A1, and A2 are also tied to logic ground.

Note: An external pull-up is required on the SDA line (open-drain output).

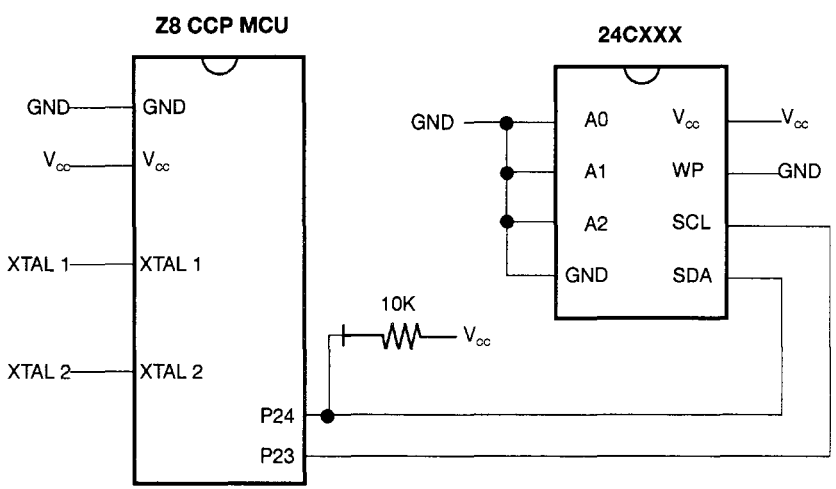


Figure 11. Generic Z8 CCP MCU-to-24CXXX Interface

ADDITIONAL REFERENCES

Appendix Summary

Refer to Appendix A for the 24CXXX Interface Software Flowchart and accompanying subroutine flowcharts.

Refer to Appendix B for applicable sample source code. The program listed in Appendix B will write/read the EEPROM in the stand-alone mode shown in Figure 11. The Z8 CCP MCU was clocked at 4 MHz for this sample code.

Note: Ensure that the selected EEPROM set-up and hold times are met at the application's Z8 clock frequency.

Secondary Sources

Atmel Nonvolatile Memory Data Book, Atmel Corporation, 1996.

I²C Bus Specifications, Philips Semiconductors Microcontroller Data Handbook, Signetics Corp, 1992.

APPENDIX A: Z8 CCP MCU 24CXXX—INTERFACE SOFTWARE FLOWCHARTS

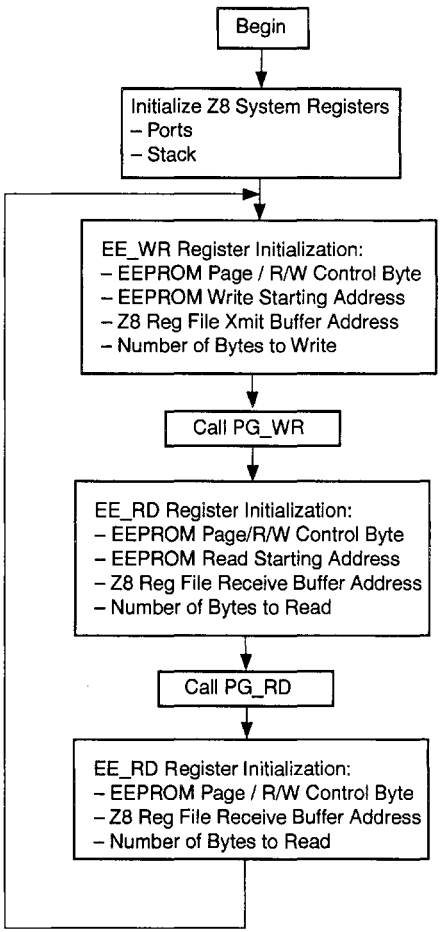

Begin
Initialize Z8 System Registers
– Ports
– Stack
EE_WR Register Initialization:
– EEPROM Page / R/W Control Byte
– EEPROM Write Starting Address
– Z8 Reg File Xmit Buffer Address
– Number of Bytes to Write
Call PG_WR
EE_RD Register Initialization:
– EEPROM Page/R/W Control Byte
– EEPROM Read Starting Address
– Z8 Reg File Receive Buffer Address
– Number of Bytes to Read
Call PG_RD
EE_RD Register Initialization:
– EEPROM Page / R/W Control Byte
– Z8 Reg File Receive Buffer Address
– Number of Bytes to Read

Figure 12. Z8 CCP MCU—24CXXX Interface Software Flowchart

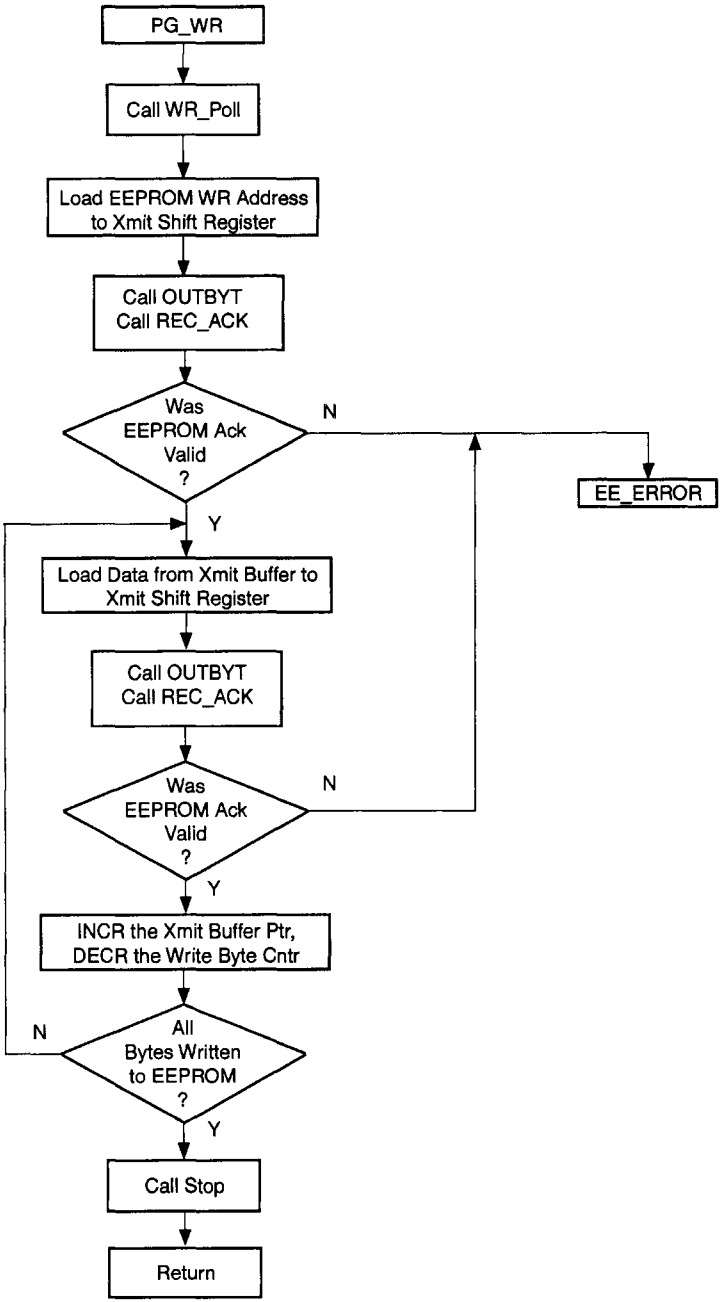


Figure 13. PG_WR Subroutine Flowchart

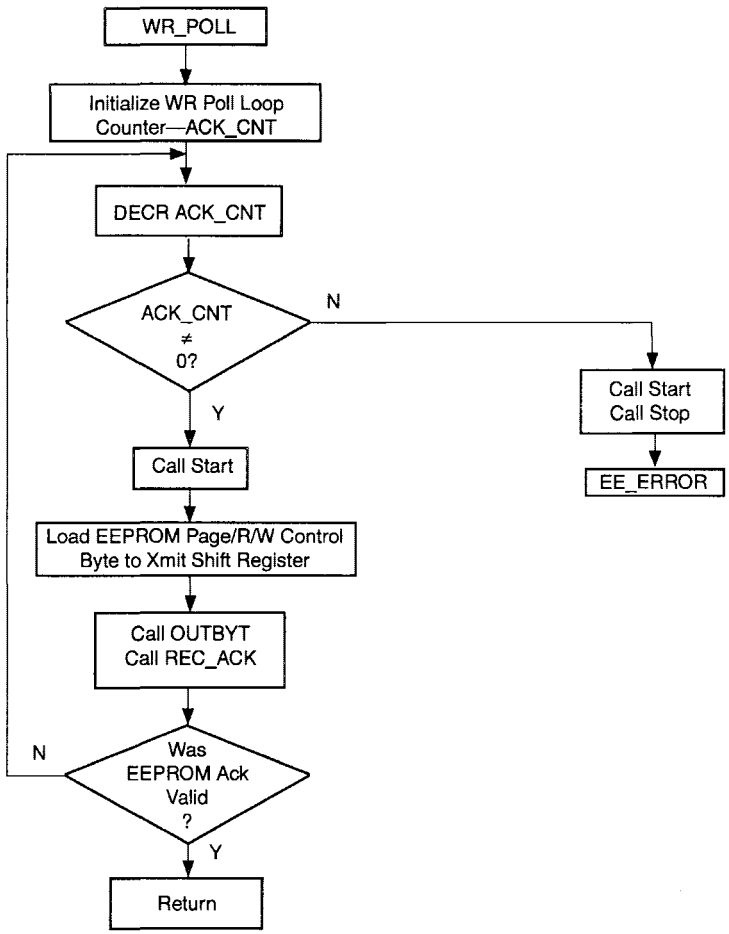


Figure 14. WR_Poll Subroutine Flowchart

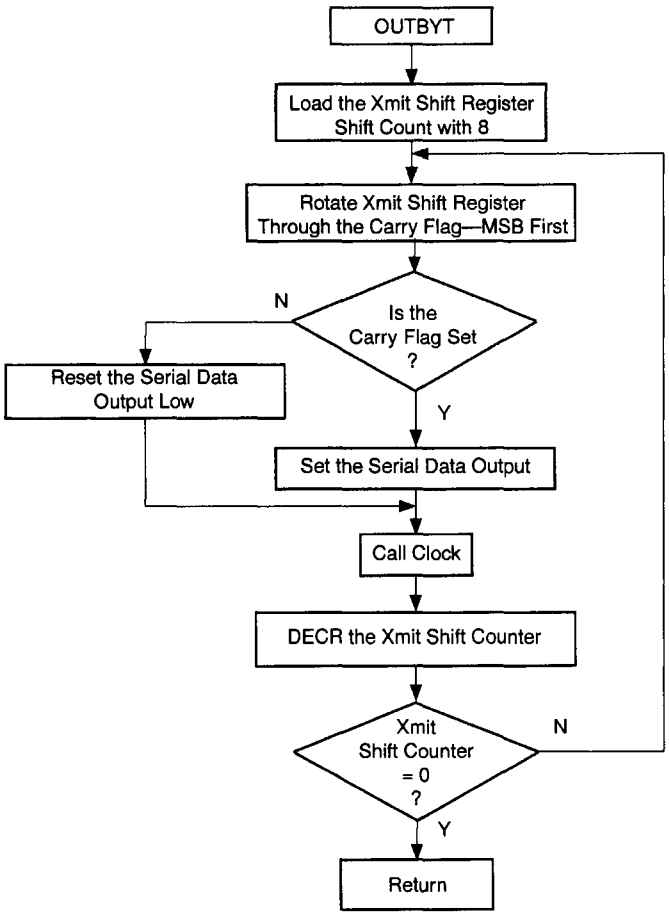


Figure 15. OUTBYT Subroutine Flowchart

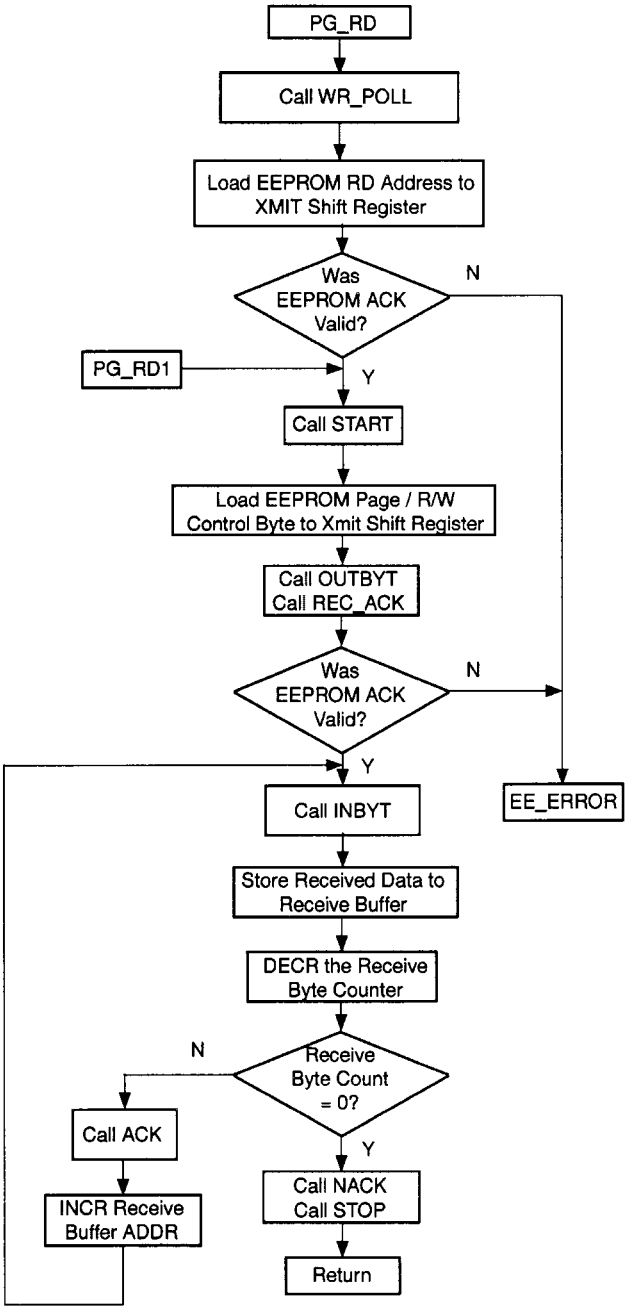


Figure 16. PG_RD Subroutine Flowchart

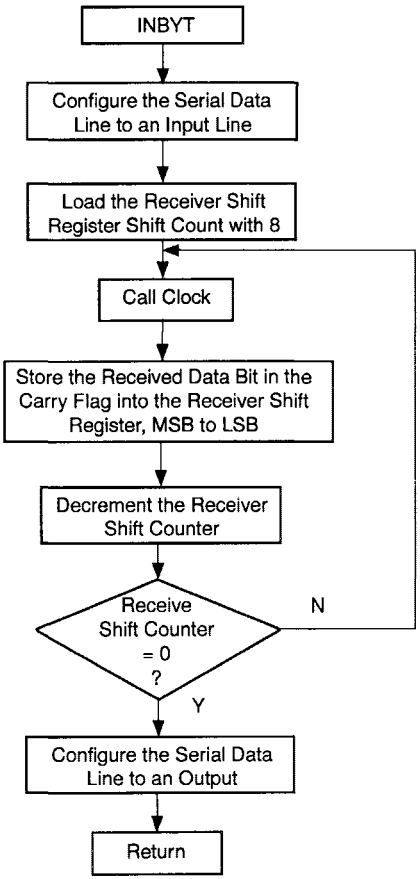

INBYT
Configure the Serial Data Line to an Input Line
Load the Receiver Shift Register Shift Count with 8
Call Clock
Store the Received Data Bit in the Carry Flag into the Receiver Shift Register, MSB to LSB
Decrement the Receiver Shift Counter
Receive Shift Counter = 0 ?
N
Y
Configure the Serial Data Line to an Output
Return

Figure 17. INBYT Subroutine Flowchart

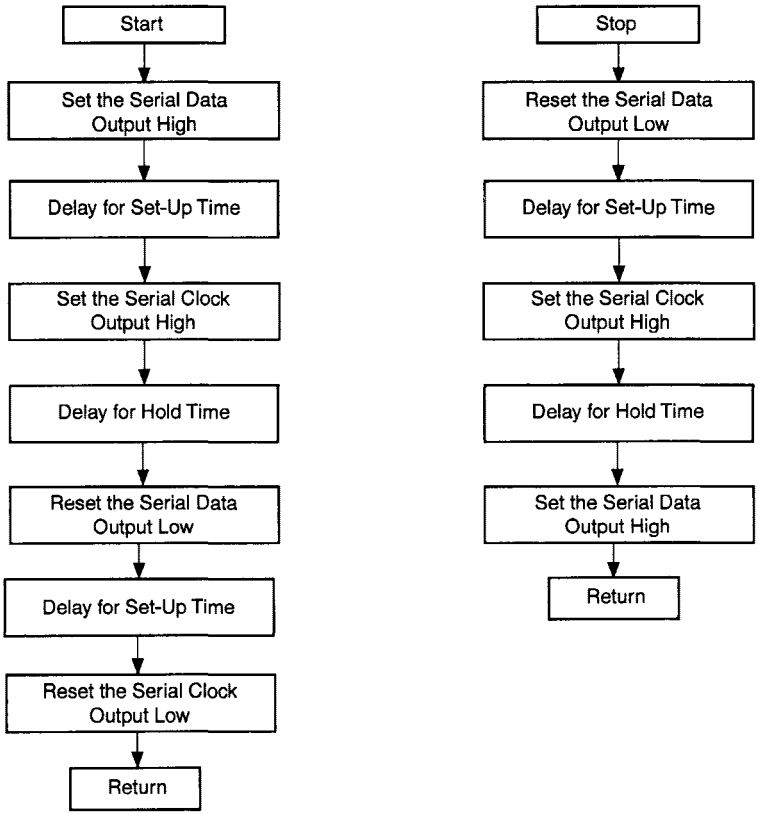

Start
Set the Serial Data Output High
Delay for Set-Up Time
Set the Serial Clock Output High
Delay for Hold Time
Reset the Serial Data Output Low
Delay for Set-Up Time
Reset the Serial Clock Output Low
Return
Stop
Reset the Serial Data Output Low
Delay for Set-Up Time
Set the Serial Clock Output High
Delay for Hold Time
Set the Serial Data Output High
Return

Figure 18. Start and Stop Subroutine Flowcharts

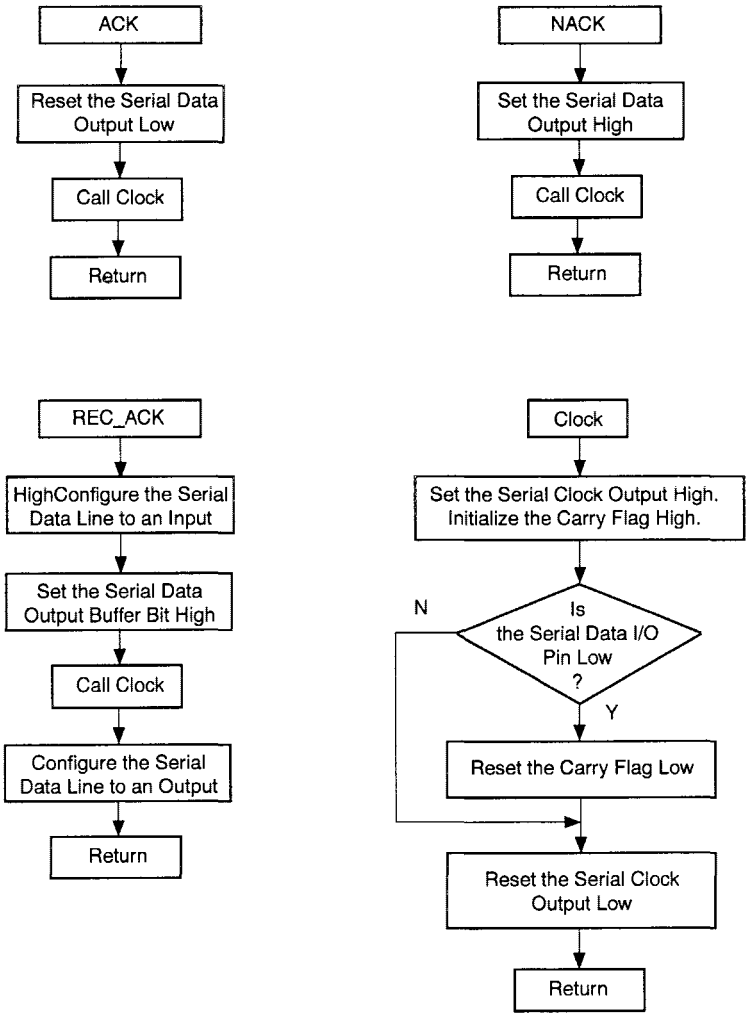


Figure 19. ACK, REC_ACK, NACK, and Clock Subroutine Flowcharts

APPENDIX B: Z8 MCU-TO-24CXXX INTERFACE SOFTWARE LISTING

Applicable sample source code is shown on the following pages.

```
;*******************************************************************************
;*        TITLE:       Z8 CCP MCU - 24Cxxx Interface Software                  *
;*        REVISION:    V1.00                                                   *
;*                                                                             *
;*                                                                             *
;*******************************************************************************
;
;         Z86Cxx MCU I/O DEFINITION:
;
;                - P2.4  =  24Cxx SERIAL DATA (SDA) I/O PIN
;                - P2.3  =  24Cxx SERIAL CLOCK (SCL) INPUT
;
;
;THIS IS A STAND ALONE PROGRAM WHICH DEMONSTRATES THE GENERIC Z8 CCP MCU TO
;24Cxxx INTERFACE SOFTWARE.  THE Z8 SUPPORTS EEPROM PAGE WRITE AND PAGE READ
;OPERATIONS (PAGE SIZE OF 1-16 BYTES).  IN THIS EXAMPLE, THE DATA STORED IN
;REGISTER FILE BANK 40H (THE DESIGNATED I2C TRANSMIT BUFFER) IS WRITTEN TO THE
;FIRST 16 BYTES OF THE EEPROM IN A SINGLE 16 BYTE PAGE WRITE OPERATION.  THIS
;EEPROM DATA IS THEN READ BACK, 8 BYTES AT A TIME, AND STORED IN REGISTER FILE
;BANK 50H (THE DESIGNATED I2C RECEIVE BUFFER).  THE DATA IS READ USING TWO
;EEPROM READ TECHNIQUES.  THE FIRST 8 BYTES ARE READ USING THE RANDOM ADDRESS
;SEQUENTIAL READ MODE WHILE THE NEXT 8 BYTES ARE READ USING THE SEQUENTIAL
;CURRENT ADDRESS READ MODE.
;
;
;*******************************************************************************
;*                          REGISTER EQUATE TABLE                              *
;*******************************************************************************
;
;REGISTER BANK 60H USED FOR I2C CONTROL REGISTERS
;
DATA        .EQU    060H            ;SERIAL I/O DATA HOLDING REGISTER
VALUE       .EQU    061H            ;DATA OUT/COMPARE REGISTER
TEMP        .EQU    062H            ;GENERAL PURPOSE TEMP REG
BIT_CNT     .EQU    063H            ;SERIAL TRANSFER BIT LOOP COUNTER
ACK_CNT     .EQU    064H            ;EEPROM WR ACK POLLING COUNTER
EE_PTR      .EQU    065H            ;BYTE ADDRESS POINTER TO EEPROM
RAM_PTR     .EQU    066H            ;ADDRESS POINTER TO Z8 REGISTER FILE
DEV_ADDR    .EQU    067H            ;24C01 Ax/Px DEVICE ADDRESS/CONTROL BYTE
OLD_P2M     .EQU    067H            ;P2M IMAGE REGISTER
BYTE_CNT    .EQU    068H            ;BYTE COUNT FOR EEPROM PAGE TRANSFERS
;
;*******************************************************************************
;*                          EEPROM INTERFACE MASK EQUATES                      *
;*******************************************************************************
;
EE_START    .EQU    #00H            ;EEPROM START ADDRESS (00H)
XRAM_START  .EQU    #40H            ;REG BANK 40H = I2C XMIT BUFFER
RRAM_START  .EQU    #50H            ;REG BANK 50H = I2C REC BUFFER (START)
RRAM_START1 .EQU    #58H            ;REG BANK 58H = I2C REC BUFFER (MIDDLE)
SDA_HI      .EQU    #10H            ;EEPROM DATA BIT HI - (P24)
SDA_LO      .EQU    #0EFH           ;EEPROM DATA BIT LO - (P24)
SCLK_HI     .EQU    #08H            ;EEPROM CLK BIT HI  - (P23)
SCLK_LO     .EQU    #0F7H           ;EEPROM CLK BIT LO  - (P23)
CODE        .EQU    #0F0H           ;INITIAL EEPROM CODE BYTE
NUMBER_8    .EQU    #08H            ;#BYTES TO PAGE RD (8 BYTE PAGE)
NUMBER_16   .EQU    #10H            ;#BYTES TO PAGE WR (16 BYTE PAGE)
WR_CNTRL0   .EQU    #0A0H           ;ADDRESS/PAGE 0 EEPROM WR CONTROL
;
;NOTE: THE SDA_xx AND SCLK_xx EQUATES REFLECT THE USE OF Z8 I/O P24 AND P23.
```

```
;       IF DIFFERENT I/O PINS ARE USED, THESE EQUATES MUST BE CHANGED TO
;       REFLECT THE NEW I/O BIT SELECTIONS.
;
;*****************************************************************************
;*                          INTERRUPT VECTOR TABLE (0000H)                   *
;*****************************************************************************
;
          .WORD  BEGIN                ;INT VECTOR TO RE-START
          .WORD  BEGIN
          .WORD  BEGIN
          .WORD  BEGIN
          .WORD  BEGIN
          .WORD  BEGIN
;
;*****************************************************************************
;*                          SYSTEM INITIALIZATION                            *
;*****************************************************************************
;
BEGIN:    DI                          ;START AT 000CH, DISABLE INTERRUPTS
          LD     P01M, #04H           ;04 = INT STACK, P0 = OUTPUTS
          LD     P3M, #01H            ;P2 OUT = P/PULL
          LD     P2, #0FFH            ;INITIALIZE P2 OUTPUTS HI (xxx1 1xxx)
          LD     OLD_P2M, #00H        ;IMAGE OF CURRENT P2M REG (xxx0 0xxx)
          LD     P2M, OLD_P2M         ;PORT 2 = OUTPUTS (xxx0 0xxx)
          LD     SPL, #80H            ;INITIALIZE STACK TO BANK 80H, PICK THE
                                      ;HIGHEST BANK AVAILABLE FOR Z8 MCU USED
;
;THE MINIMUM SYSTEM INTIALIZATION CODE FOR THIS APPLICATION IS NOW COMPLETE.
;
;*****************************************************************************
;*        EE_WR:  THE FOLLOWING INSTRUCTIONS PERFORM THE EEPROM WRITE        *
;*        OPERATION.  THREE POINTERS AND A TRANSFER COUNT MUST BE SET UP BY  *
;*        THE USER PRIOR TO CALLING THE PG_WR SUBROUTINE:                    *
;*                                                                           *
;*        - DEV_ADDR = 24Cxx Ax/Px address with the R/W bit reset to 0,      *
;*        - EE_PTR   = Target EEPROM address ptr to start data storage at,   *
;*        - RAM_PTR  = Z8 reg file buffer ptr to start data storage from,    *
;*        - BYTE_CNT = # bytes to xfer (1 to max page buffer number).        *
;*                                                                           *
;*        IN THIS EXAMPLE, THE DATA STORED IN REGISTER FILE LOCATIONS 40H-   *
;*        4FH (THE DEFINED I2C TRANSMIT BUFFER) IS READ OUT AND WRITTEN TO   *
;*        EEPROM LOCATIONS 00H-0FH OF BANK 0 (A 16 BYTE PAGE WRITE).        *
;*****************************************************************************
;
EE_WR:    LD     DEV_ADDR, WR_CNTRL0  ;INIT EEPROM DEVICE ADDR/PAGE/WRITE BITS
          LD     EE_PTR, EE_START     ;INIT EEPROM BYTE ADDRESS POINTER
          LD     RAM_PTR, XRAM_START  ;INIT REG FILE WRITE BUFFER POINTER
          LD     BYTE_CNT, NUMBER_16  ;INIT EEPROM PAGE TRANSFER BYTE CNTR
          CALL   PG_WR                ;WRITE "NUMBER" OF BYTES TO EEPROM
;
;*****************************************************************************
;*        EE_RD:  THE FOLLOWING INSTRUCTIONS PERFORM THE EEPROM RANDOM       *
;*        ADDRESS SEQUENTIAL READ OPERATION.  THREE POINTERS AND A TRANSFER  *
;*        COUNT MUST BE SET UP BY THE USER PRIOR TO CALLING THE PG_RD        *
;*        SUBROUTINE:                                                        *
;*                                                                           *
;*        - DEV_ADDR = 24Cxx Ax/Px address with the R/W bit reset to 0,      *
;*        - EE_PTR   = Target EEPROM address ptr to start data storage at,   *
;*        - RAM_PTR  = Z8 reg file buffer ptr to start data storage from,    *
;*        - BYTE_CNT = # bytes to xfer (1 to max page buffer number).        *
```

```
;*                                                                          *
;*        THE ABOVE FOUR BYTES ARE REQUIRED FOR A RANDOM READ (BYTE_CNT =1)  *
;*        AND A RANDOM SEQUENTIAL READ (BYTE_CNT >1). THE EEPROM READ IS     *
;*        ACCOMPLISHED BY CALLING THE PG_RD ROUTINE.                         *
;*                                                                          *
;*        IN THIS EXAMPLE, EEPROM ADDRESSES 00H-07H ARE READ OUT AND STORED  *
;*        TO REGISTER FILE LOCATIONS (50H-57H -- THE FIRST HALF OF THE I2C   *
;*        RECEIVE BUFFER).                                                   *
;*                                                                          *
;*        FOR A CURRENT ADDRESS READ OR A CURRENT ADDRESS SEQUENTIAL READ,
;*        THE EE_PTR REGISTER IS NOT USED.  THIS TYPE OF EEPROM READ IS      *
;*        EXECUTED BY CALLING THE PG_RD1 ROUTINE - SEE EE_RD'.               *
;****************************************************************************
;
EE_RD:    LD      DEV_ADDR, WR_CNTRL0   ;INIT EEPROM DEVICE ADDR/PAGE/WRITE BITS
          LD      EE_PTR, EE_START      ;INIT EEPROM BYTE ADDRESS POINTER
          LD      RAM_PTR, RRAM_START   ;INIT REG FILE READ BUFFER POINTER
          LD      BYTE_CNT, NUMBER_8    ;INIT EEPROM SEQ TRANSFER BYTE CNTR
          CALL    PG_RD                 ;READ "NUMBER" OF BYTES FROM EEPROM
;
;****************************************************************************
;*        EE_RD': THE FOLLOWING INSTRUCTIONS PERFORM THE CURRENT ADDRESS     *
;*        SEQUENTIAL READ OPERATION.  TWO POINTERS AND A TRANSFER COUNT MUST *
;*        BE SET UP BY THE USER PRIOR TO CALLING THE PG_RD1 SUBROUTINE:      *
;*                                                                          *
;*        - DEV_ADDR = 24Cxx Ax/Px address with the R/W bit reset to 0,      *
;*        - RAM_PTR  = Z8 reg file buffer ptr to start data storage from,    *
;*        - BYTE_CNT = # bytes to xfer (1 to max page buffer number).        *
;*                                                                          *
;*        FOR A CURRENT ADDRESS READ OR A CURRENT ADDRESS SEQUENTIAL READ,   *
;*        THE EE_PTR REGISTER IS NOT USED (THE INTERNAL EEPROM ADDRESS       *
;*        POINTER IS USED).  THIS TYPE OF EEPROM READ IS EXECUTED BY CALLING *
;*        THE PG_RD1 ROUTINE.                                                *
;*                                                                          *
;*        IN THIS EXAMPLE, EEPROM ADDRESSES 08H-0FH ARE READ OUT AND STORED  *
;*        TO REGISTER FILE LOCATIONS (58H-5FH -- THE SECOND HALF OF THE I2C  *
;*        RECEIVE BUFFER).                                                   *
;****************************************************************************
;
EE_RD':   LD      DEV_ADDR, WR_CNTRL0   ;INIT EEPROM DEVICE ADDR/PAGE/WRITE BITS
          LD      RAM_PTR, RRAM_START1  ;INIT REG FILE READ BUFFER POINTER
          LD      BYTE_CNT, NUMBER_8    ;INIT EEPROM SEQ TRANSFER BYTE CNTR
          CALL    PG_RD1                ;READ "NUMBER" OF BYTES FROM EEPROM
;
;
          JP      EE_WR                 ;PROGRAM COMPLETE, RE-START THE EEPROM
                                        ;WRITE/READ CYCLE
;
;
;THE FOLLOWING ROUTINES PERFORM THE ACTUAL EEPROM BYTE WRITE, PAGE WRITE,
;RANDOM READ, CURRENT ADDRESS READ, AND SEQUENTIAL READ ONCE THE ABOVE
;DESCRIBED PARAMETERS ARE PASSED.
;
;****************************************************************************
;*        PG_WR:  This routine writes a page of data (length controlled by   *
;*        reg BYTE_CNT) using the EEPROMs page write feature.  A byte write  *
;*        accomplished by setting BYTE_CNT = 1.                              *
;****************************************************************************
;
PG_WR:    CALL    WR_POLL               ;VERIFY EEPROM READY, SEND START,WR CMND
```

```
          LD      DATA, EE_PTR          ;EEPROM WR BYTE ADDRESS
          CALL    OUTBYT                ;SEND WRITE ADDRESS DATA
          CALL    REC_ACK               ;SEARCH FOR EEPROM ACKNOWLEDGE COMMAND
          JP      C, EE_ERR             ;IF CARRY SET, NO EEPROM ACK RECEIVED
PG_WR1:   LD      DATA, @RAM_PTR        ;GET THE REG FILE DATA TO WR TO EEPROM
          CALL    OUTBYT                ;SEND WRITE DATA
          CALL    REC_ACK               ;SEARCH FOR EEPROM ACKNOWLEDGE COMMAND
          JP      C, EE_ERR             ;IF CARRY SET, NO EEPROM ACK RECEIVED
          INC     RAM_PTR               ;INC FOR NEXT REG FILE BYTE
          DEC     BYTE_CNT              ;TEST FOR ALL BYTES WRITTEN OUT
          JR      NZ, PG_WR1
          CALL    STOP                  ;PAGE WRITE COMPLETE - SEND STOP
          RET
;
;*******************************************************************************
;*        PG_RD:  This routine reads a block of data (length controlled by     *
;*        reg BYTE_CNT) by using the EEPROMs sequential read feature.  A       *
;*        single byte read is accomplished by setting BYTE_CNT = 1.            *
;*******************************************************************************
;
PG_RD:    CALL    WR_POLL               ;VERIFY EEPROM READY, SEND START,WR CMND
          LD      DATA, EE_PTR          ;EEPROM W/R START ADDRESS
          CALL    OUTBYT                ;SEND WRITE DATA
          CALL    REC_ACK               ;SEARCH FOR EEPROM ACKNOWLEDGE COMMAND
          JP      C, EE_ERR             ;IF CARRY SET, NO EEPROM ACK RECEIVED
;
;THE "DUMMY" WRITE IS COMPLETE - PROCEED WITH READ ADDRESS PARAMETERS.
;
;NOTE: THIS IS ALSO THE START POINT FOR A CURRENT ADDRESS READ OPERATION (USER
;MUST INSURE THAT A EEPROM WRITE CYCLE IS NOT PENDING WHEN A CURRENT READ
;OPERATION IS INITIATED DIRECTLY).
;
PG_RD1:   CALL    START                 ;SEND THE START COMMAND
          OR      DEV_ADDR, #01H        ;SET THE R/W BIT TO 1 - READ COMMAND
          LD      DATA, DEV_ADDR        ;SETUP EEPROM READ CONTROL
          CALL    OUTBYT                ;SEND DATA
          CALL    REC_ACK               ;SEARCH FOR EEPROM ACKNOWLEDGE COMMAND
          JP      C, EE_ERR             ;IF CARRY SET, NO EEPROM ACK RECEIVED
PG_RD2:   CALL    INBYT                 ;READ BYTE FROM EEPROM
          LD      @RAM_PTR, DATA        ;STORE EEPROM DATA TO REGISTER FILE
          DEC     BYTE_CNT              ;DEC BYTE COUNT, IS PAGE READ COMPLETE
          JR      Z, PG_RD3
          CALL    ACK                   ;SEND THE PAGE READ BYTE ACK
          INC     RAM_PTR               ;INC THE REG FILE POINTER
          JR      PG_RD2
;
PG_RD3:   CALL    NACK                  ;PAGE READ DONE, ISSUE NACK/STOP CMNDS
          CALL    STOP
          RET
;
;*******************************************************************************
;*        INBYT: SHIFT IN 8 BITS FROM THE EEPROM TO THE DATA REGISTER.         *
;*******************************************************************************
;
INBYT:    OR      OLD_P2M, SDA_HI       ;CHANGE SDA TO AN INPUT
          LD      P2M, OLD_P2M          ;UPDATE P2M
          LD      BIT_CNT, #08H         ;LOAD UP FOR 8 SHIFT TIMES
IN_1:     CALL    CLOCK                 ;CLOCK DATA
          RLC     DATA                  ;CONSTRUCT SERIAL DATA FROM EEPROM
          DEC     BIT_CNT               ;LOOP FOR 8 BIT TIMES
```

```
          JR      NZ, IN_1
          AND     OLD_P2M, SDA_LO       ;CHANGE SDA TO AN OUTPUT
          LD      P2M, OLD_P2M          ;UPDATE P2M
          RET
;
;*****************************************************************************
;*        OUTBYT: SHIFT OUT 8 BITS FROM THE DATA REGISTER TO THE EEPROM.     *
;*****************************************************************************
;
OUTBYT:   LD      BIT_CNT, #08H         ;LOAD UP FOR 8 SHIFT TIMES
OUT_1:    RLC     DATA                  ;ROTATE BIT TO TRANSMIT INTO CARRY FLAG
          JR      C, OUT_2              ;SEND CARRY STATE TO SDA
          AND     P2, SDA_LO            ;WRITE 0 TO SDA
          JR      OUT_3
OUT_2:    OR      P2, SDA_HI            ;WRITE 1 TO SDA
OUT_3:    CALL    CLOCK                 ;CLOCK OUT THIS DATA BIT
          DEC     BIT_CNT               ;LOOP TILL ALL 8 BITS SEND
          JR      NZ, OUT_1
          RET
;
;*****************************************************************************
;*        REC_ACK: TEST FOR THE EEPROM RECEIVE ACKNOWLEDGE (RESULT IN CF)    *
;*****************************************************************************
;
REC_ACK:  OR      OLD_P2M, SDA_HI       ;CHANGE SDA TO AN INPUT
          LD      P2M, OLD_P2M          ;UPDATE P2M
          OR      P2, SDA_HI
          CALL    CLOCK                 ;GENERATE A CLOCK PULSE
          AND     OLD_P2M, SDA_LO       ;CHANGE SDA TO AN OUTPUT
          LD      P2M, OLD_P2M          ;UPDATE P2M
          RET
;
;*****************************************************************************
;*        WR_POLL:  POLL THE EEPROM TO DETERMINE WHEN THE ACTUAL NV WRITE    *
;*        CYCLE IS COMPLETE.  THE ROUTINE TESTS 256 TIMES FOR A VALID EEPROM *
;*        ACK WHICH IS INDICATED BY A RESET CARRY FLAG UPON RETURN.          *
;*****************************************************************************
;
WR_POLL:  LD      ACK_CNT, #00H         ;MAX NUMBER OF TIMES TO POLL IS 256
WR_POLL1: DEC     ACK_CNT
          JR      Z, WR_POLL2
          CALL    START                 ;SET-UP TO ACCESS THE EEPROM
          LD      DATA, DEV_ADDR        ;SET-UP EEPROM DEVICE ADDRESS/PAGE
          CALL    OUTBYT                ;SEND WRITE DATA
          CALL    REC_ACK               ;SEARCH FOR EEPROM ACKNOWLEDGE COMMAND
          JR      C, WR_POLL1           ;LOOP AND RE-TEST IF NO ACK RECEIVED
          RET
WR_POLL2: CALL    START                 ;ISSUE A START COMMAND
          CALL    STOP                  ;ISSUE A STOP/LOW PWR COMMAND
          JP      EE_ERR                ;NO EEPROM ACK RECEIVED - JP TO ERROR
;
;*****************************************************************************
;*        START:  ISSUE A START COMMAND.                                     *
;*****************************************************************************
;
START:    OR      P2, SDA_HI            ;SEND START CONDITION - SDA SET TO 1
          NOP                           ;GIVE HOLD TIME IN CASE OF FAST OSC
          NOP
          OR      P2, SCLK_HI           ;SET CLOCK OUTPUT HIGH
          NOP                           ;GIVE SET-UP TIME IN CASE OF FAST OSC
```

```
        NOP
        AND     P2, SDA_LO          ;RESET SDA TO 0
        NOP                         ;GIVE HOLD TIME IN CASE OF FAST OSC
        NOP
        AND     P2, SCLK_LO         ;SET CLOCK OUTPUT LO
        RET
;
;*****************************************************************************
;*      STOP:  ISSUE A STOP COMMAND                                          *
;*****************************************************************************
;
STOP:   AND     P2, SDA_LO          ;WRITE SDA TO 0 - STOP CONDITION
        NOP                         ;GIVE HOLD TIME IN CASE OF FAST OSC
        NOP
        OR      P2, SCLK_HI         ;SET CLOCK OUTPUT HI
        NOP                         ;GIVE SET-UP TIME IN CASE OF FAST OSC
        NOP
        OR      P2, SDA_HI          ;WRITE SDA TO 1
        RET
;
;*****************************************************************************
;*      ACK: ISSUE A ACK - CONTINUE A PAGE READ                              *
;*****************************************************************************
;
ACK:    AND     P2, SDA_LO          ;RESET SDA TO 0
        CALL    CLOCK               ;GENERATE A CLOCK PULSE
        RET
;
;*****************************************************************************
;*      NACK:  ISSUE A NACK - TERMINATE A BYTE READ.                         *
;*****************************************************************************
;
NACK:   OR      P2, SDA_HI          ;SET SDA TO 1
        CALL    CLOCK               ;GENERATE A CLOCK PULSE
        RET
;
;*****************************************************************************
;*      CLOCK:  ISSUE A CLOCK PULSE - SDA LEVEL STORED IN CARRY FLAG.        *
;*****************************************************************************
;
CLOCK:  OR      P2, SCLK_HI         ;SET THE CLOCK LINE HI
        SCF                         ;INIT CARRY FLAG FOR NO EEPROM ACK,
                                    ;PROVIDE HDWR SETUP TIME BEFORE BIT READ
        TM      P2, SDA_HI          ;TEST THE SDA INPUT FOR A LOGIC 0
        JR      NZ, CLOCK_1         ;JUMP IF SDA HIGH, NO EEPROM ACK (CF=1)
        RCF                         ;EEPROM ACK RECEIVED, RESET THE CF = 0
CLOCK_1: AND    P2, SCLK_LO         ;RESET THE CLOCK LINE LO
        RET
;
;*****************************************************************************
;*      EE_ERR:  This routine performs the system shutdown procedure         *
;*      when the EEPROM does not respond to a write/read request.            *
;*****************************************************************************
;
EE_ERR: JP      EE_ERR              ;PUT YOUR ERROR HANDLER HERE
;
        .END
```

USING SOFTWARE DESIGN TECHNIQUES TO MAXIMIZE Z8 SYSTEM NOISE IMMUNITY

BY EXTENDING TYPICAL HARDWARE METHODOLOGIES, SEVERAL SOFTWARE TECHNIQUES CAN BE EMPLOYED TO FURTHER PROTECT AGAINST NOISE UPSETS.

INTRODUCTION

Many articles have been written on the hardware aspects of MCU-based system design to maximize noise immunity, covering such topics as PCB layout, component selection, and circuit protection devices. Unfortunately, the role software design can play is often neglected. Good software design can help a system survive upsets caused by noise that circumvents hardware protection schemes. This application note provides an introduction to a variety of software techniques that can provide a Z8®- MCU-based system with improved noise immunity and reliability of operation.

FAULT-AVOIDANCE AND FAULT-TOLERANT DESIGN

Hardware Design Methodologies

Most product designers think of fault avoidance and fault tolerance in terms of hardware system redundancies. The rationale is to prevent improper system actions in the event of external and internal influences encountered. External influences include supply voltage and frequency variations as well as EMC events (voltage surges, fast transients, electromagnetic radiation, and electrostatic discharge) and climatic conditions (short- and long-term temperature and humidity affects). Internal influences include systematic and random errors. Systematic or design errors (as they are more commonly known) occur due to the complexity of the product under design. They can only be solved through re-design efforts. Despite diligence in component selection and product construction, random errors and failures cannot be completely eliminated. The system designer typically uses Failure Mode Effect Analysis (FMEA) or Fault -Tree Analysis (FTA) techniques to identify deficiencies in end-product performance when various components randomly fail. These operational deficiencies may then be addressed and corrected.

Extending Design Methodologies to Software

Typical hardware design methodologies may be extended to software. Elimination of systematic errors is a primary goal here due to the MCU system software's necessarily unique nature. After a thorough analysis of the end-product performance requirements, there are several methods that can be employed in the software design and test phases. These include the following:

Writing Structured Code. This includes dividing all system tasks into software blocks for individual testing. Each block should have a single entry, a normal exit point, a fault exit point (if required), and a clear specification of data on which the block may execute.

Full-Time System Control. Software-controlled hardware functions should be in a known inactive state during program self test, initialization, and fault-detected lock-out modes.

Full Documentation. Well-structured, thorough documentation greatly improves prospects of eliminating systematic errors through individual and peer reviews, as well as ensuring that the implemented functions fulfill the original design goals.

Testing Methodologies. To detect systematic errors, input and path testing techniques help ensure that module input conditions and program loops are exercised.

MCU system software also can be designed for improved fault tolerance through the addition of on-line fault detection safeguards. These safeguards provide an interlock between system hardware and software operation. When faults are detected, the user program can determine the severity of the fault and take the defined course of proper operation. This operation would include action to be taken (record the error, prompt the end user, shut down and lock out the system, and so on) as well as the speed of implementing the corrective action.

The concept of fault-tolerant software has and is being promulgated by several standards agencies (Ex. UL 1998 Table 7 parameters) for products that control potentially dangerous operations, such as a gas burner control. These standards specify safety-critical areas of operations, which dictate the need for primary and secondary safeguard systems (hardware and software).

The software methods discussed in this application note focus on the three MCU areas where errors occur or are most often induced: MCU memory, I/O, and program flow. Graceful recovery or a controlled system lock-out from any externally induced fault or random component failure is the system design goal.

IMPLEMENTING FAULT TOLERANCE WITH THE Z8

MCU Memory

The Z8 CCP™ MCU has internal ROM and register file RAM memory. A variety of ROM and RAM checking methods may be employed to validate memory data and check for stuck bits as well as single- and double-bit errors. The techniques used include checksums, parity bits, CRC, and comparison of redundant memory segments. These memory tests are usually conducted and completed at power-up, but may also be run continuously as part of the normal program scan. In this latter case, partial memory tests are usually conducted during each scan so that minimum MCU processing time is lost.

A sample Z8® MCU ROM check routine is provided in Appendix A: Sample Z8 Programs, Listing 1. (Refer to Figure 1: ROM Check Program Flowchart, which follows.) This subroutine computes and verifies a 16-bit ROM checksum, and provides for an entire ROM memory check at power-up as well as a continuous back-ground ROM check (256 bytes/program scan). As seen from the sample code, the architecture of the Z8 MCU makes ROM testing easy via use of the LDC instruction. This instruction allows ROM data to be directly accessed by the user program and operated on accordingly. (This also makes ROM look-up tables easy to use.)

Note: ROM tests may be conducted in Z8 CCP MCUs (user program in the internal device ROM) even when the ROM Protect mask option is enabled.

A detected and verified ROM fault is a "hard" or fatal system failure, which should result in the termination of module operation.

A sample Z8 register file RAM check routine is provided in Appendix A: Sample Z8 Programs, Listing 2. (Refer to Figure 2: RAM Check Program Flowchart, which follows.) This routine tests the register file by writing and then verifying alternating byte patterns to all general purpose register locations. Thus, all address and data bit and byte combinations are exercised. The RAM check finishes by writing and verifying that all locations have been cleared to zero. The RAM check is designed as a one-time test to be called during the system power-up initialization. The routine could be modified to perform partial register file checks continuously per program scan (as in the previous ROM test example). An on-going checksum test could also be computed on the entire register file with constant updates for every data write. A simpler approach is to store all critical RAM data in redundant locations. Then, as part of the system test procedures called each program scan, these memory locations can be compared.

For example, assume there are 32 critical-data parameters present in a user program. The data is stored in Z8 register file banks 1 and 2 (locations 10H through 2FH). Fault-tolerant design techniques can be implemented by storing the complement of this data in another pair of register banks (mirror image data at register file banks 4 and 5, locations 40H through 5FH). At the end of each program scan, the register file banks are compared for matching values. The Z8 instruction set supports this type of data storage and comparison through the complement (COM), exclusive-or (XOR), and loop count instruction—decrement and jump if not zero (DJNZ). A sample bank-checking-code sequence for this example is shown below:

```
     LD     R5, #10H          ;Set up primary data bank reg ptr
     LD     R6, #40H          ;Set up complimentary data bank reg ptr
     LD     R7, #20H          ;Test 32 bytes of data
LOOP:LD     R8, @R6           ;Load the compliment data
     XOR    R8, @R5           ;Test primary and compliment
     JP     Z, RAM_FAULT      ;XOR = 0; No data match--RAM error
     DJNZ   R7, LOOP          ;Test all 32 bytes for correct compliment
```

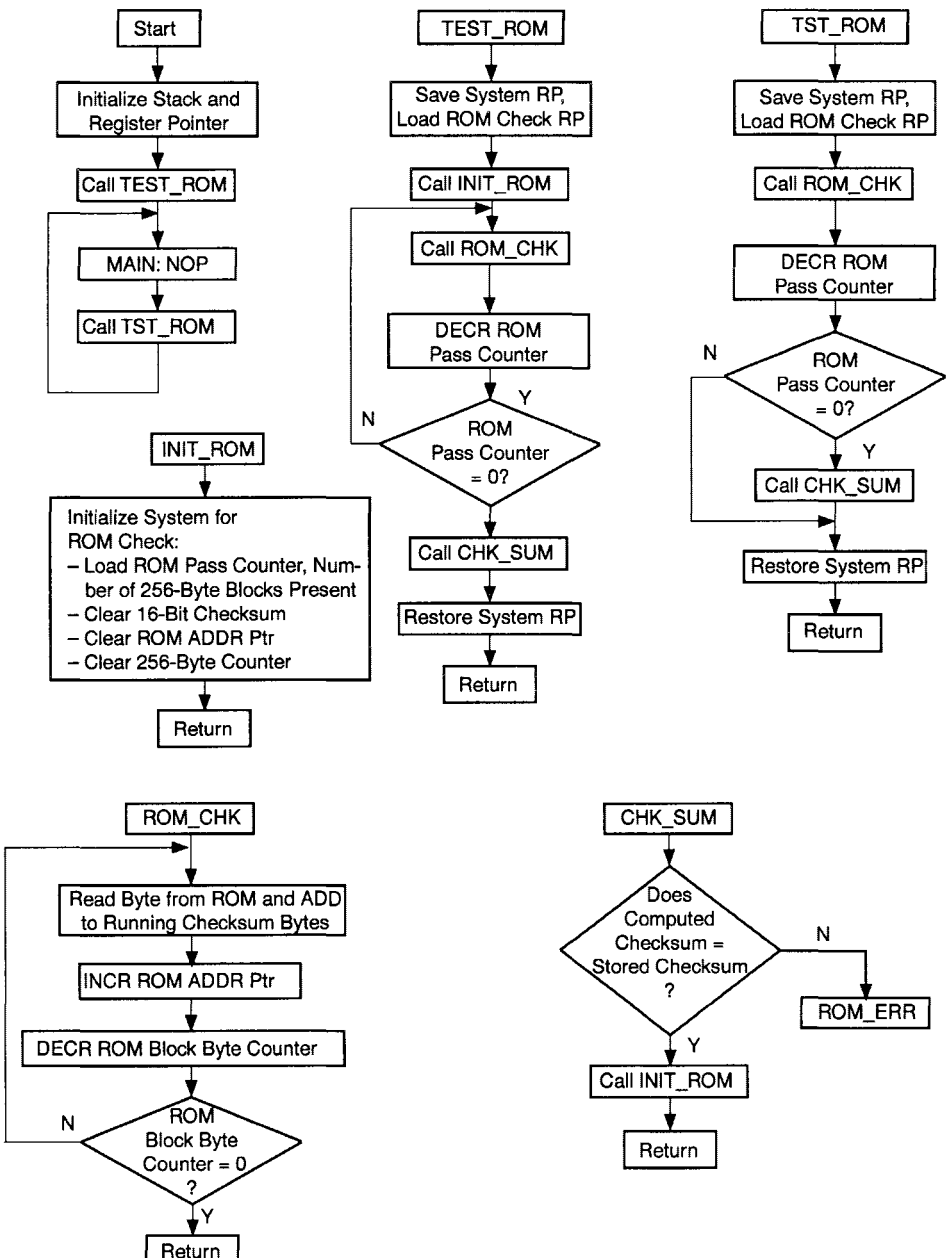


Figure 1. ROM Check Program Flowchart

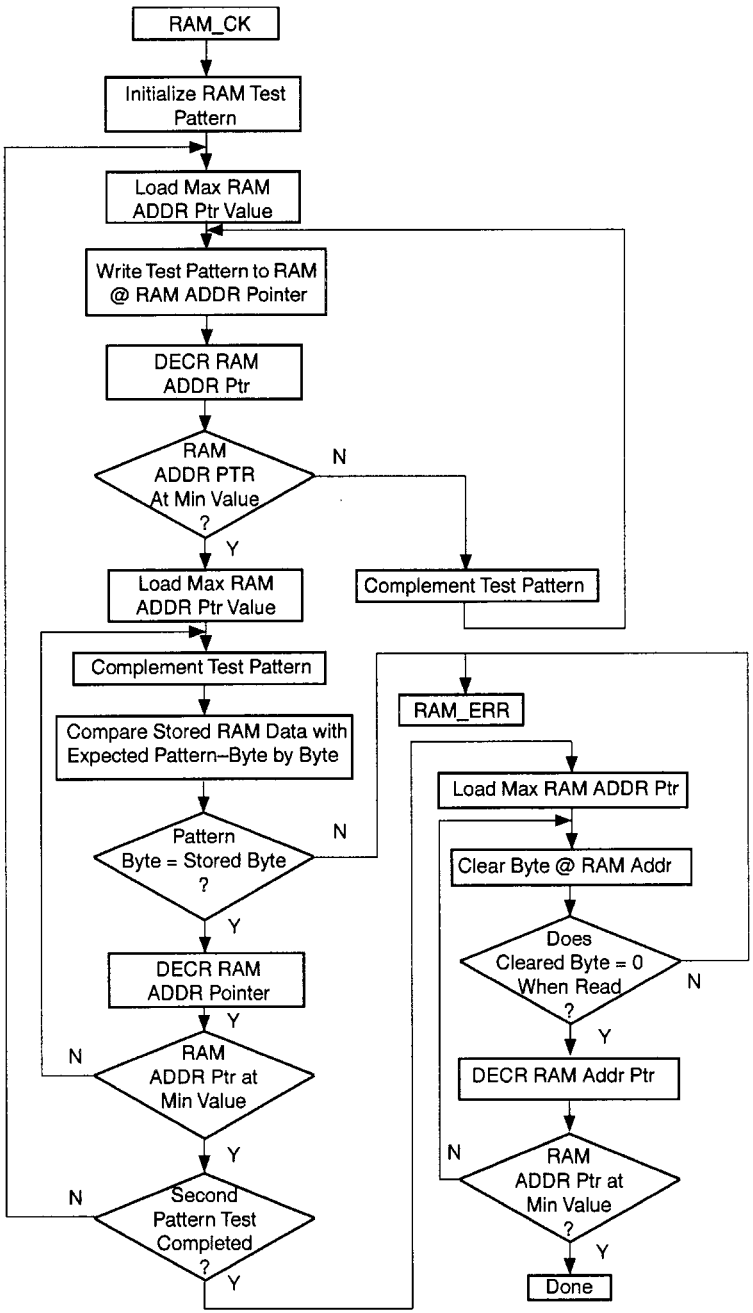


Figure 2. RAM Check Program Flowchart

A detected and verified RAM error is usually a "soft" error. The recovery procedure typically validates that the RAM location(s) in question is still operating correctly (by performing RAM pattern testing) and then restoring the data to the parameter's default or current state value.

MCU I/O

MCU I/O is often subject to external influences despite hardware protection. Software techniques like input debouncing and I/O validity and range checking are available to ensure system I/O integrity. Fault tolerance may be improved by the creation and maintenance of a redundant and complementary I/O image table within the Z8® register file.

Software input debouncing is analogous to hardware input debouncing and keeps out noise induced transients from being seen by the user program as actual input state changes.

There are many different ways to implement input debouncing. The simplest technique is to simply count like-kind consecutive input states and, when the specified number is achieved, interpret that as a valid state. The problem with this method is that valid state changes can be adversely delayed by repetitive noise conditions. An improved debounce method is to take a continuous signal level weighted average to determine the valid input state. This method assigns a debounce counter for each input port bit to be debounced. A low- and high- level threshold count is set for the user application with the difference between levels being the degree of hysterisis. Appendix A: Sample Z8 Programs, Listing 3 shows this type of debounce method. (Also, refer to Figure 3: Input Debounce Program Flowchart.)

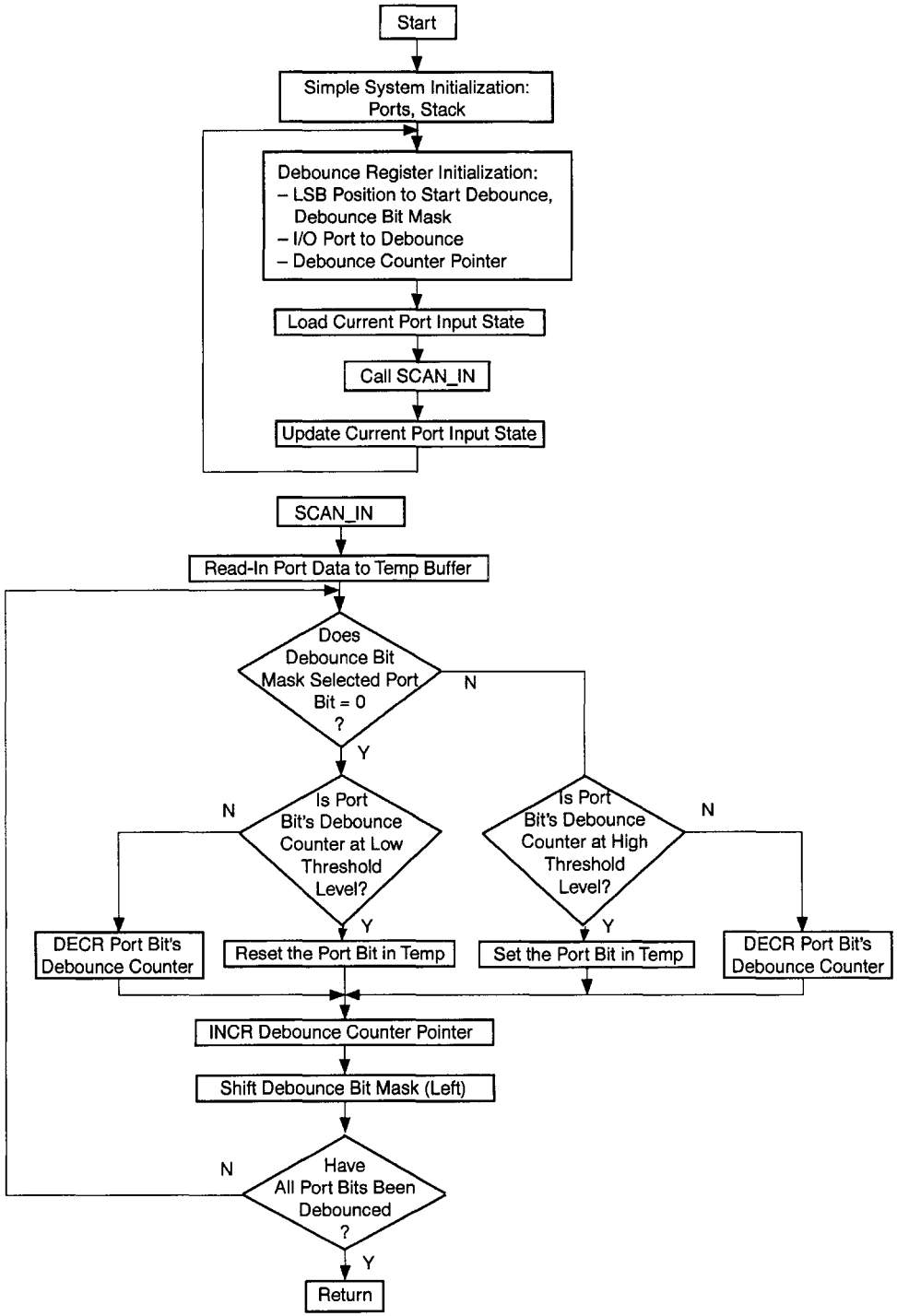

Start
Simple System Initialization: Ports, Stack
Debounce Register Initialization:
– LSB Position to Start Debounce, Debounce Bit Mask
– I/O Port to Debounce
– Debounce Counter Pointer
Load Current Port Input State
Call SCAN_IN
Update Current Port Input State
SCAN_IN
Read-In Port Data to Temp Buffer
Does Debounce Bit Mask Selected Port Bit = 0 ?
N
Y
Is Port Bit's Debounce Counter at Low Threshold Level?
N
Y
Is Port Bit's Debounce Counter at High Threshold Level?
N
Y
DECR Port Bit's Debounce Counter
Reset the Port Bit in Temp
Set the Port Bit in Temp
DECR Port Bit's Debounce Counter
INCR Debounce Counter Pointer
Shift Debounce Bit Mask (Left)
Have All Port Bits Been Debounced ?
N
Y
Return

Figure 3. Input Debounce Program Flowchart

In the Z8, the I/O port registers are mapped into locations 00H-03H of the register file. MCU fault tolerance may be enhanced by constructing a redundant store of these registers.

I/O image tables allow for redundancy checking on the debounced input states the user program makes its' logical decisions on. The input debounce counters (see Appendix A: Sample Z8 Programs, Listing 3) may be compared to the corresponding input port image table bit for logic state consensus. If a discrepancy is found, recovery options may include changing the input to its reset default state.

They also provide for output state checking. As shown in the block diagram of Figure 4, each Z8 bidirectional I/O port has an input register and buffer, an output register and buffer, and associated control logic. Writing to the I/O port register causes data to be stored in the port's output register.

This feature allows output port data to be initialized prior to the I/O pins being configured as outputs. Reading a port register causes the data on the external I/O port pins to be read whether defined as inputs or outputs.

The output data read-in is compared to the output data of the I/O image table. If the output data matches, the outputs are being properly represented on the external interface and there are no shorts or stuck pins. Note that this type of output validation checking can only be done on outputs configured for push/pull active drive.

Finally, I/O Image Tables Allow the "Safe" Use of Z8 Boolean Instructions for I/O Z8 boolean instructions (example: TM, TCM, AND, OR, XOR, COM) operate on a read-modify-write basis. If the user program operates on an I/O port directly with a boolean instruction, there is the potential danger of a noise pulse corrupting a bidirectional I/O bit (see the previous description of the Z8's bidirectional port structure). It is, therefore, safer to perform all user program I/O modifications on the I/O port image table registers and then write the I/O ports directly via load instructions (LD). (Refer to Figure 4.)

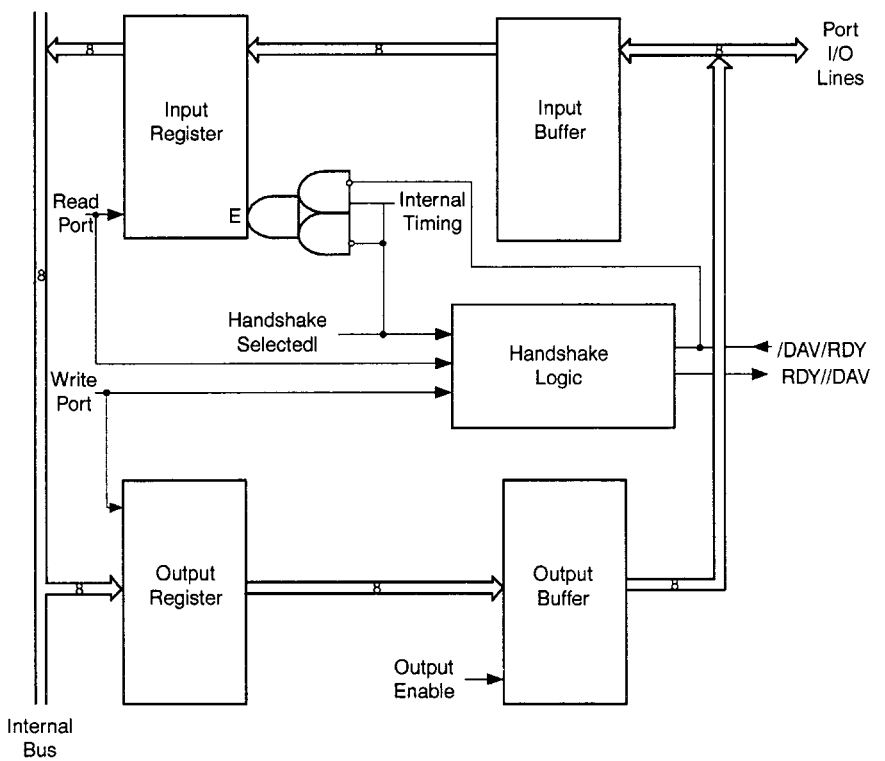


Figure 4. Ports 0, 1, 2 Generic Block Diagram

MCU Program Flow

Normal MCU program flow also may be interrupted by external influences despite hardware protection. Typical fault modes here are the altering of the MCU program counter (causing the software to jump to random locations) and a noisy reset line (which can lock-up MCU operation). There are several hardware and software techniques that can be embedded in the design to help a system recover from these type of faults.

Hardware Methods

The Z8 CCP MCU family provides on-chip hardware resources to aid in EMI upset recovery. A primary feature is the elimination of the external reset line in favor of internal hardware low-voltage/brown-out detection and reset control. This feature eliminates the possibility of improper MCU operation or lock up due to a noisy or glitching reset input that would violate MCU reset input timing specifications. On-chip, low-voltage detection guarantees the Z8 to be in a known operating state (RUN, HALT, STOP, or RESET) during V_{CC} transients. This feature eliminates the need for a reset input and its associated external support and conditioning circuitry on the Z8. When in the reset mode, all Z8 bidirectional I/O's are configured as high-impedance inputs so no output loads can be driven. This is the same I/O condition present after reset release during the power-up initialization. Z8 outputs are enabled only after the I/O direction bits are configured in the I/O port mode control registers.

The Z8's on-chip timer/counters may be used to validate MCU system timing. A Z8 timer may be used to periodically measure the time duration of a known applied external signal such as a fixed input clock frequency, 60 Hz AC line (zero cross detect), and so on. When the external signal is in a valid range, the user program knows the MCU program time base (clock source) is within design specification.

Also available on the Z8 CCP MCU is an on-chip Watch-Dog Timer (WDT) circuit. A WDT is a hardware circuit that counts input pulses. If it is allowed to count a pre-determined maximum number of counts, it generates a hardware timeout signal that can be used as a fault lock-out signal or master system reset. When operating normally, the user program will re-trigger or reset the WDT count to its initial value before the count expires and generates the system reset. Thus, there is a hardware safeguard (WDT) on system software operation.

The most reliable WDT would have the following operational properties. The WDT clock source must be reliable and MCU system-clock hardware independent. The WDT should offer timing flexibility to match the user application by having a time-out period consistent with the process being controlled (not too fast or too slow). Since the WDT is a hardware safeguard on controlled software action, the WDT must remain as independent to the software as possible. Thus, the WDT should be initiated by application of system power and armed independently of software that may not run properly at start-up. Similarly, software should not be allowed to disable an operating WDT. "Kicking" or re-triggering the WDT (resetting the timer count to its initial value) should be through a unique software command. Finally, the WDT time-out should generate a "safe" system reset condition.

While some applications mandate the use of an external WDT—to force safety-critical hardware to an "off" state—an MCU's well-designed, on-chip WDT will suffice for many applications. The Z8 CCP MCU's internal WDT provides maximum user flexibility and system protection by having the following design properties:

- Internal RC clock source, independent of the MCU clock source
- Automatic WDT start-up at power-on reset (feature enabled via a ROM mask or OTP programming option bit)
- Once enabled, the WDT cannot be disabled by software
- Unique WDT refresh opcode
- Software-programmable modes of operation via the access protected WDTMR register:
  - WDT on in any/all MCU operating modes: RUN, HALT, STOP
  - Programmable WDT time-out periods (5–100 ms @ 5V V_{CC})

In the Z8 CCP MCU, WDT operation options are controlled by the WDT Mode Register (WDTMR). This register may be written by the user program only during the first 64 system-clock cycles after a POR, WDT reset, or Stop-Mode Recovery. This access protection guards against rogue writes that might happen with program counter corruption. It controls time-out period selection and WDT activity in the various MCU operating modes.

Although well-designed, the Z8's on-chip WDT will provide maximum protection only if properly used. The WDT is re-triggered by software execution of the WDT instruction (5FH opcode). For maximum effectiveness, the user program should only re-trigger the WDT at one point in the main program. This is usually done when all software system checks have been validated. This concept will be discussed fully in the "Software Methods" section, which follows.

Finally, all Z8 CCP MCU I/O pins have internal input protection diodes to the high and low side rails. This

provides effective device protection against voltage transients with the addition of current-limiting resistors.

Software Methods

In addition to the hardware safeguards on program flow, user software should be written to actively detect signs of a system fault/upset. Software for detection of MCU memory and I/O faults has already been reviewed. Similar software techniques may be applied to prevent and detect illegal program flow. The simplest prevention method is to fill any unused ROM program memory with the Z8's NOP opcode (FFH) up to the last three program bytes—where a jump to start instruction is located.

Example:

```
...     (last program byte)
NOP
.
.
NOP
JP   START
```

In this example, START is the label for program address 000CH (starting program address location for all Z8 MCUs). Now, if the program counter (PC) was corrupted and execution re-directed to the unused memory area, the Z8's PC would re-sync on the NOP instructions and jump to the start location.

Note that the Z8's PC will automatically wrap around when the maximum address value for the device is reached. Similarly, all unused Z8 interrupt sources should have their jump vectors pointed to a "bad interrupt" system recovery routine. This routine will flush the interrupt condition by restoring the flag register and stack pointer, and direct the PC to continue at the user desired location.

Another prevention measure is to examine the user program hex file and eliminate all Z8 MCU operating mode opcodes that appear as part of the normal program. The critical opcodes here are for the HALT (7FH), STOP (6FH), and WDT (5FH) instructions. For example, the instruction LD R5,#6FH loads the immediate value of 6FH into working register 5. This provides a machine code sequence of 5C 6F. If a PC disruption occurs, the potential exists for the PC to start execution with the 6F opcode, and thus execute a STOP instruction. For this reason, these critical opcodes should only exist in one place: the actual routine where a power-down mode is entered or the WDT is kicked. These power-down routines should incorporate software checks to validate their execution and be externally guardbanded by the Z8's WDT.

There are several software methods to detect program flow disruption. Consider the simplified user program flow shown in Figure 5.

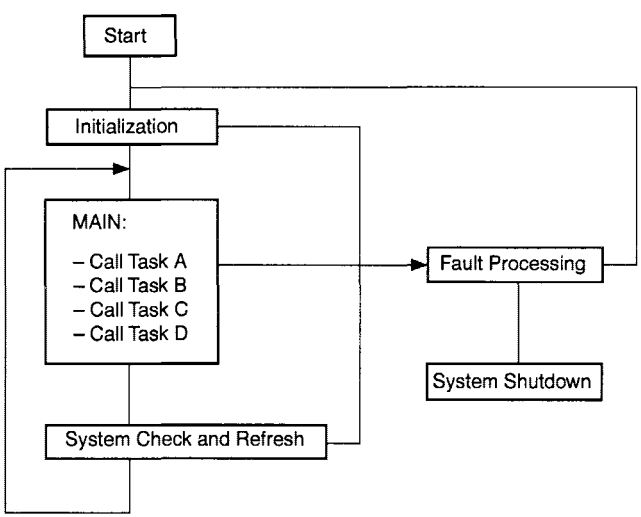


Figure 5. Simplified Z8 User Program Flow

Proper main program flow executes tasks in the A - B - C - D order. By using sequence numbers that are unique, pre-assigned numbers for each subroutine, system software may be used to validate proper program flow. In this example, the main program will set the initial sequence number then call task A. Task A when starting, running, and/or concluding execution, will validate that its sequence number matches the supervisory sequence number. If the sequence numbers do not match, program execution will be directed to the fault handler. This process continues as the main program sets the predetermined sequence numbers for each succeeding task called.

Additional fault checking can be carried out at the task level. Each task should monitor its passed input parameters for validity. Time-slot monitoring is a technique that uses the Z8 MCU timer resource to measure each task's execution duration. Execution time outside the acceptable minimum/maximum timing window can be signaled as an operating fault. Another method involves intra-task logical flow monitoring (program sequencing discussed previously is an inter-task logical flow monitoring technique). Checkpoint flags can be set up inside each task block to monitor that all critical functions inside a task are executed. At task end, the checkpoint flags, a unique one for each task function, are checked to see that each critical function ran once, and only once. A simplified and less precise implementation makes use of a flowcheck counter. When each critical task function is completed, the flowcheck counter is incremented. At the end of the task, the flowcheck counter is tested for the proper count. If checkpoint flags are not set correctly or flowcheck counts don't match, it indicates program flow in the task was upset and critical functions were either not executed or repeated execution. Program action is then directed to the fault handler software. Note that these same methods can be used in interrupt service routines to insure that they are not being called too frequently (or infrequently) relative to main program operation.

Continuing with this example, when the main loop completes execution, the system check and refresh routine runs. This subroutine performs the global system checks, such as the background ROM check, critical-redundant RAM checks, system time-base check, final sequence code check, and so on. If all global checks pass, the system is deemed to be in control and system refresh commences. Refresh consists of updating/re-writing the I/O ports, the I/O port mode control registers, system control registers (interrupt and stack facilities), and timer control registers. The purpose of this refresh is to ensure that any noise corrupted system control registers are re-initialized to the required state. Finally, the WDT instruction is executed—the only place the WDT instruction is executed.

When a fault is detected by any of the above discussed software procedures, the user program jumps to the fault handling routine. This subroutine will determine the fault root cause and take the user defined action. The action taken will depend if the error is considered fatal (results in system lock-out) or recoverable. (Try a system re-start.) Of course it is up to the system designer to decide what a fatal and recoverable error is for each end product. The fault handler makes use of the sequence code and checkpoint flags to determine the actual root-cause failure mode. Thus, sequence numbers and checkpoint flags serve a secondary role as fault "tracing codes" that indicate where proper system operation was terminated.

When the user program ends up vectoring back to the program start location (whether by WDT timeout, bad interrupt vector, or by action of the fault handler), the user's initialization sequence must account for this type of re-start. This is typically done by having the system initialization code determine if a "warm" or "cold" start condition exists. A cold start is defined as an initial power-up sequence. For this case, the complete system check and initialization to system default state must be performed. A warm start is defined as a system that has been running, but for some reason (system upset/ fault detection), is attempting a re-start. Instead of defaulting to the cold start reset sequence, the system checks to see if it is possible to carry on program activity from the point of the upset (or as close as possible to it). Getting back to the example shown in Figure 5, the redundant register banks holding all critical program and I/O data would be tested. If the data checks out as valid, the system can jump to the system check and refresh routine and continue on. To the product's end user, the system glitch goes unnoticed, as system operation is ongoing.

CONCLUSION

This application note focused on software methods available to enhance a Z8 MCU-based system's immunity to EMC upset. Topics ranging from improved documentation and testing practices to specific techniques for preventing and detecting faults in MCU memory, I/O, and program flow were reviewed. These methods have been developed to meet the standards required for safety-critical products.

These same techniques may be used—as MCU resources allow—in any embedded control design to enhance the end product's reliability. Appendix A contains some sample Z8 programs for internal ROM and RAM checking and input debouncing. Appendix B contains the "Z8 CCP MCU System Designer's Checklist," a list of do's and don'ts for the Z8-based system designer.

REFERENCES

Handbook of EU EMC Compliance, 1995 Edition, Compliance Design, Inc.

UL 1998 Standard for Safety-Related Software Underwriters Laboratories, Inc., 1994.

"Watchdog Timer Techniques," Embedded Systems Programming, April, 1995.

APPENDIX A: SAMPLE Z8 PROGRAMS

Listing 1. Sample Z8 ROM Check

```
;*******************************************************************************
;*      TITLE:ROM_CHECK                                                        *
;*      REVISION: V1.00                                                        *
;*      DATE: 8/23/95                                                          *
;*                                                                             *
;*******************************************************************************
;
;ROM_CHECK: A stand-alone Z8 MCU program that demonstrates a full-scale ROM check
;          as well as an on-line ROM check (256 bytes checked per user program
;          scan). All ROM check registers used are defined in register file bank 1.
;
;*******************************************************************************
;*      WORKING REGISTER EQUATE TABLE                                          *
;*******************************************************************************
;
ROM_PTR      .EQU   RR0          ;Working Rpr - 16-bit ROM Address Ptr
ROM_PTR_HI   .EQU   R0           ;Working Reg - ROM Addr Ptr High
ROM_PTR_LO   .EQU   R1           ;Working Reg - ROM Addr Ptr Low
BYTE_CNT     .EQU   R2           ;Working Reg - ROM Check Byte Cntr
;
;
ROM_PASS     .EQU   R3           ;Working Reg - On-line ROM Block Cntr
ROM_CHAR     .EQU   R4           ;Working Reg - ROM Chk Summing Register
CHKSUM_LO    .EQU   R5           ;Working Reg - Computed ROM Chksum (LO)
CHKSUM_HI    .EQU   R6           ;Working Reg - Computed ROM Chksum (HI)
;
;*******************************************************************************
;*      ROM CHECK CONSTANT EQUATES                                             *
;*******************************************************************************
;
ROM_LO       .EQU #0FFH          ;Stored ROM Checksum Lo-Byte Value
ROM_HI       .EQU #0B5H          ;Stored ROM Checksum Hi-Byte Value
ROM_SIZE     .EQU #10H           ;# Of 256-Byte Pages In Z8 MCUs ROM—a 4K ROM
                                 ;Is Used In This Sample Prgm
```

```
;
;*******************************************************************************
;*      INTERRUPT VECTOR TABLE (0000H)                                         *
;*******************************************************************************
;
.WORD  START
.WORD  START
.WORD  START
.WORD  START
.WORD  START
.WORD  START
;
;*******************************************************************************
;*      SYSTEM INITIALIZATION                                                  *
;*******************************************************************************
;
START:  DI                      ;Start At 000CH, Disable Interrupts
        LD     P01M,#04H        ;Configure Z8 for Internal Stack
        LD     SPL,#80H         ;Initialize SP to Top of the Reg File
        SRP    #00H             ;Set Default Register Pointer to Bank 0
;
;Simple initialization required for ROM_CHECK code to run is
;complete. At initialization, The full ROM check is run.
;
CALL TEST_ROM                   ;Call for full ROM check
;
;
MAIN:   NOP                     ;MAIN substitutes for the typical user program
        NOP
;
;When The user program scan (MAIN) is complete, the On-line System Diagnostic
;Routines are called before starting the next program scan at MAIN.
;
        CALL  TST_ROM           ;On-line ROM Test (256 Bytes/Prgm Scan)
;
                                ;All other system diagnostic/refresh routines
                                ;are called before jumping to start the next
                                ;user program scan.
;
        JP MAIN
```

```
;
;*****************************************************************************
;*      TEST_ROM: THE FOLLOWING SUBROUTINE PERFORMS THE INITIAL SYSTEM ROM     *
;*      CHECK. THE ROM TEST USES THE PREDEFINED REGISTERS OF REG FILE BANK 1   *
;*      AND CHECKS THE ENTIRE ROM IN ONE PASS. WHEN ALL BLOCKS OF THE ROM HAVE *
;*      BEEN SUMMED, THE STORED 16-BIT CHECKSUM IS COMPARED TO THE COMPUTED    *
;*      VALUE. IF THE CHECKSUMS DO NOT MATCH, PROGRAM EXECUTION JUMPS TO THE   *
;*      IF THE CHECKSUMS DO NOT MATCH, PROGRAM EXECUTION JUMPS TO ROM ERROR    *
;*      ROUTINE. OTHERWISE, THE ON-LINE ROM CHECK REGISTERS ARE RE-INITIALIZED *
;*      AND TESTING GOES ON. ROM TESTING IS SIMPLIFIED IN THE Z8 THROUGH USE OF*
;*      THE LOAD CONTROL INSTRUCTION (LDC), WHICH ALLOWS THE Z8 ROM CONTENTS TO*
;*      BE READ INTO THE REGISTER FILE.                                        *
;*****************************************************************************
;
TEST_ROM:   PUSH    RP                  ;Save System REGPTR (RP)
            SRP     #10H                ;Get The Reserved ROM Check Reg Bank
            CALL    INIT_ROM            ;Init Reg Bank 1 For ROM Check
TEST_1:     CALL    ROM_CHK             ;Computes CHK_SUM for 256-Byte Block
            DJNZ    ROM_PASS, TEST_1    ;Test For All ROM Blocks Checked
            CALL    CHK_SUM             ;If All Checked, Call The Chksum Test
            POP     RP                  ;ROM OK, Restore RP,Continue Program
            RET                         ;Run

;*****************************************************************************
;*      TST_ROM: THE FOLLOWING SUBROUTINE PERFORMS THE ON-LINE RUNNING ROM CHECK.*
;*      THE ROM TEST USES THE PREDEFINED REGISTERS OF REG FILE BANK 1 AND CHECKS*
;*      256-BYTE ROM BLOCKS PER CALL/PROGRAM SCAN. WHEN ALL BLOCKS OF THE ROM  *
;*      HAVE BEEN SUMMED, THE STORED 16-BIT CHECKSUM IS COMPARED TO THE COMPUTED*
;*      VALUE. IF THE CHECKSUMS DO NOT MATCH, PROGRAM EXECUTION JUMPS TO THE   *
;*      ROM ERROR ROUTINE. OTHERWISE,THE ON-LINE ROM CHECK REGISTERS ARE       *
;*      RE-INITIALIZED AND TESTING GOES ON. ROM TESTING IS SIMPLIFIED IN       *
;*      THE Z8 THROUGH USE OF THE LOAD CONTROL INSTRUCTION (LDC), WHICH ALLOWS *
;*      THE Z8 ROM CONTENTS TO BEREAD INTO THE REGISTER FILE.                  *
;*****************************************************************************
```

```
;*******************************************************************************
;
TST_ROM     PUSH    RP                  ;Save System RP
            SRP     #10H                ;Get The Reserved ROM Check Reg Bank
            CALL    ROM_CHK             ;Computes CHK_SUM For 256-byte Block
            DJNZ    ROM_PASS, TST_1     ;Test For All ROM Blocks Checked
CALL        CHK_SUM                     ;IF All Checked, Call The Chksum Test
TST_1:      POP     RP                  ;ROM OK, Restore RP,Continue Program Run
            RET
;
;*******************************************************************************
;*      INIT_ROM: THE FOLLOWING SUBROUTINE INITIALIZES THE REGISTERS THAT      *
;*      THAT HAVE BEEN RESERVED IN REGISTER BANK 1 FOR THE POWER-UP AND RUN    *
;*      TIME ON-LINE ROM CHECKS.                                               *
;*******************************************************************************
;
INIT_ROM:   LD      ROM_PASS,ROM_SIZE   ;Counts Number Of 256-byte ROM Checks
                                        ;4K ROM = 10H ROM Passes
            CLR     CHKSUM_LO;          ;Init Running Checksum Low Byte
            CLR     CHKSUM_HI           ;Init Running Checksum High Byte
            CLR     ROM_PTR_LO          ;Init ROM Addr Pointer - Low Byte
            CLR     ROM_PTR_HI          ;Init ROM Addr Pointer - High Byte
            CLR     BYTE_CNT            ;Init Byte Cntr For 256-Byte ROM Pass
            RET
;
;*******************************************************************************
;*      ROM_CHK:COMPUTES THE RUNNING CHECKSUM IN 256-BYTE BLOCKS.THE COMPUTED  *
;*      VALUE IS STORED IN CHKSUM_LO AND CHKSUM_HI.                            *
;*******************************************************************************
;
ROM_CHK:    LDC     ROM_CHAR, @ROM_PTR  ;Load The ROM Byte To Reg File
            ADD     CHKSUM_LO, ROM_CHAR ;Compute Running Checksum
            JR      NC, ROM_1
            INC     CHKSUM_HI           ;Fix For Addition Carry, 16-Bit Result
ROM_1:      INCW    ROM_PTR             ;Do Next ROM Byte
            DJNZ    BYTE_CNT, ROM_CHK   ;Test For End Of 256-Byte Test Block
RET
```

```
;
;*******************************************************************************
;*      CHK_SUM: COMPARES THE COMPUTED ROM CHECKSUM TO THE STORED VALUES.       *
;*      A FAILED CHECKSUM MATCH CAUSES PROGRAM EXECUTION TO JUMP TO THE         *
;*      ROM FAULT HANDLER.                                                      *
;*******************************************************************************
;
CHK_SUM:   CP      CHKSUM_LO, #ROM_LO  ;All ROM Locations Read, Test Chksum Lo
           JR      NE, ROM_ERR         ;Bad ROM Checksum Is A Fatal Error
           CP      CHKSUM_HI, #ROM_HI  ;All ROM Locations Read, Test Chksum Hi
           JR      NE, ROM_ERR         ;Bad ROM Checksum Is A Fatal Error
           CALL    INIT_ROM            ;Chksums Good, Set-up Run Time ROM Chk
RET
;
;*******************************************************************************
;*         ROM_ERR: USER-DEFINED ACTION FOR SOFTWARE DETECTED ROM ERROR.*
;*******************************************************************************
;
ROM_ERR:   NOP                         ;USER DEFINED ROM ERROR FAULT HANDLER
           JP      ROM_ERR
;
;*******************************************************************************
;*         CHECKSUM ADD BYTE: DEFINE BYTE VALUE FOR CHECKSUM COMPUTATION*
;*******************************************************************************
;
;ROM_HI and ROM_LO are the stored 16-bit program checksum. The checksum
;add byte is used to make the ROM_LO byte equal to FFH. A simple way to
;calculate the value of the add byte is to initialize it to 00H and the
;ROM_HI byte to FFH. An emulator is used to run this ROM check program
;and compute the initial checksum test result. The value computed for
;CHKSUM_HI is inserted as the value of ROM_LO. The checksum test is
;repeated and the checksum add byte's final value is the difference
;needed to make the CHKSUM_LO byte value equal to FFH once the ROM_HI
;byte is set.
           .BYTE   #0CCH               ;Add Byte To Make ROM_LO = FFH
;
           .END
```

Listing 2. Sample Z8 RAM Check

```
;*******************************************************************************
;*        TITLE:        RAM_CHECK                                                *
;*        REVISION:     V1.00                                                    *
;*        DATE:         8/23/95                                                  *
;*******************************************************************************
;
;RAM_CHECK: A stand-alone Z8 MCU program that demonstrates a RAM test. This
;           program tests every general-purpose register file byte/bit to
;           ensure the respective HI/LO state is functional. This code is
;           written for a Z8 CCP MCU with a 128-byte general-purpose register
;           file. It can be easily adapted for other Z8 register file sizes
;           by altering the RAM_MAX value.
;
;*******************************************************************************
;*                      REGISTER EQUATE TABLE                                  *
;*******************************************************************************
;
RAM_VAL       .EQU    0FFH              ;RAM Test Pattern Byte (GPR)
RAM_PTR       .EQU    0FEH              ;RAM Test Pointer (SPL)
;
;*******************************************************************************
;*                      RAM CHK CONSTANT EQUATES                               *
;*******************************************************************************
;
RAM_MAX      .EQU    #7FH               ;Top Of 128-byte GP Reg File
PATTERN      .EQU    #55H               ;Initial Test Pattern Byte
COM_PATTERN .EQU    #0AAH               ;Complemented Initial Pattern Byte
;
;*******************************************************************************
;*                      INTERRUPT VECTOR TABLE (0000H)                         *
;*******************************************************************************
;
          .WORD  START
          .WORD  START
          .WORD  START
          .WORD  START
          .WORD  START
          .WORD  START
```

```
;
;*****************************************************************************
;*                        SYSTEM INITIALIZATION                              *
;*****************************************************************************
;
START:    DI                          ;Start At 000CH, Disable Interrupts
;
;Simple Initialization Required For RAM_CHECK Code To Run Is Complete.
;
;*****************************************************************************
;*        RAM_CHECK: The following routine performs a RAM check on the       *
;*        entire general purpose (GP) Z8 register file. The test consists    *
;*        of writing all GP registers with an alternating-byte test pattern,*
;*        verifying the pattern, then repeating the entire procedure writing*
;*        and reading the complemented pattern.                              *
;*                                                                           *
;*        Registers Used By RAM_CHECK:                                       *
;*                                                                           *
;*        SFR R254 - SPL: Defined As RAM_VAL - Holds The Test Byte           *
;*        SFR R255 - GPR: Defined As RAM_PTR - Reg File Address Pointer      *
;*                                                                           *
;*        User Defines The Following Constants For The Application:          *
;*                                                                           *
;*        RAM_MAX: Highest GP Reg File Address Available For Z8 Selected     *
;*        PATTERN: Value Of User-Defined Test Pattern Byte                   *
;*        COM_PATTERN: Complemented Value Of User-Defined Test Pattern Byte  *
;*****************************************************************************
;
RAM_CHK:  LD      RAM_VAL, PATTERN      ;Load RAM Check Pattern Value
RAM_WR:   LD      RAM_PTR, RAM_MAX      ;Set RAM Ptr To The Top Of The Reg File
RAM_WR_1: LD      @RAM_PTR, RAM_VAL     ;Store RAM Pattern
          DEC     RAM_PTR
          CP      RAM_PTR, #03H         ;All Reg File Bytes (EX I/O) Written?
          JR      EQ, RAM_RD
          COM     RAM_VAL               ;Alternate Pattern On Successive Bytes
          JR      RAM_WR_1
;
;GP Reg File Written With Alternating Test/Complemented Test Bytes,
;Read Back And Verify Each Byte/bit Value Is Correct.
;
RAM_RD:   LD      RAM_PTR, RAM_MAX      ;Set RAM Ptr To The Top Of The Reg File
RAM_RD_1: COM     RAM_VAL               ;Re-init RAM Test Byte Starting Value
          CP      RAM_VAL, @RAM_PTR     ;Compare RAM Patterns
```

```
          JP      NE, RAM_ERR
          DEC     RAM_PTR
          CP      RAM_PTR, #03H          ;All Reg File Bytes (Ex I/O) Checked?
          JR      NE,RAM_RD_1
          CP      RAM_VAL, COM_PATTERN ;Test For First Or Second RAM Test Pass
          JR      NE, RAM_CLR            ;Jump To Clear RAM Is Second Pass Done
          JP      RAM_WR                 ;Start The Second Reg File Test Pass
;
;The RAM Pattern Testing Has Been Successfully Complete, Clear/Verify Clear
;the Entire General-Purpose Reg File
;
RAM_CLR:  LD      RAM_PTR, RAM_MAX       ;Set Ptr To The Top Of The Reg File
ZERO:     CLR     @RAM_PTR               ;Zero Out The Reg File (Locations)
          CP      @RAM_PTR, #00H         ;Verify RAM Location Is Cleared To 00H
          JP      NZ, RAM_ERR
          DEC     RAM_PTR
          CP      RAM_PTR, #03H          ;All Reg File Bytes (Ex I/O) Cleared?
          JR      NE, ZERO
;
DONE:     JP      DONE                   ;RAM Check Successfully Completed
;
;*******************************************************************************
;*        RAM_ERR:  USER-DEFINED ACTION FOR SOFTWARE DETECTED RAM ERROR.       *
;*******************************************************************************
;
RAM_ERR:  NOP                            ;User-defined RAM Error Handler
          JP      RAM_ERR
;
          .END
```

Listing 3. Sample Z8 Input Debounce

```
;*******************************************************************************
;*        TITLE:       INPUT DEBOUNCE                                          *
;*        REVISION:    V1.00                                                   *
;*        DATE:        8/23/95                                                 *
;*******************************************************************************
;
;SCAN_IN:  A stand-alone Z8 MCU program that demonstrates an input debounce
;          routine. This routine uses input state weighted averages (a simple
;          noise correction/tolerance technique) to determine the actual input
;          level. Level hysteresis is built in by using high- and low-level
;          thresholds. The debounce (threshold) level is programmable by the
;          user.
;
;*******************************************************************************
;*                       REGISTER EQUATE TABLE                                 *
;*******************************************************************************
;
IN_BUFF     .EQU   10H              ;Input Port Data To Be Debounced
BIT_MASK    .EQU   11H              ;Input Port Bit To Be Debounced
MAX_BIT     .EQU   12H              ;Port's Max Bit Position To Debounce
PORT        .EQU   13H              ;Input Port Number To Be Debounced
DB_PTR      .EQU   14H              ;Port Bit Debounce Cntr Pointer
INPUT       .EQU   15H              ;Holding Reg For Debounced Port Bits
IM_PORT0    .EQU   20H              ;I/O Port Image Table - Port 0
IM_PORT2    .EQU   22H              ;I/O Port Image Table - Port 2
;
;*******************************************************************************
;*                       INPUT DEBOUNCE CONSTANT EQUATES                       *
;*******************************************************************************
;
DB_TABLE    .EQU   #50H             ;Reg File Start Location For Deb Cntrs
HI_THRESH   .EQU   #05H             ;User-defined Deb High Threshold
LO_THRESH   .EQU   #00H             ;User-defined Deb Low Threshold
;
```

```
;*******************************************************************************
;*                          INTERRUPT VECTOR TABLE (0000H)                     *
;*******************************************************************************
;
          .WORD   START
          .WORD   START
          .WORD   START
          .WORD   START
          .WORD   START
          .WORD   START
;
;*******************************************************************************
;*                          SYSTEM INITIALIZATION                              *
;*******************************************************************************
;
START:    DI                            ;Start At 000CH, Disable Interrupts
          LD      P01M, #05H            ;05 = Int Stack, P0 = Inputs
          LD      P3M, #01H             ;P2 Out = P/Pull
          LD      P2M, #0FFH            ;Configure P2 Inputs (0111 1100)
          LD      SPL, #80H             ;Initialize Stack To Bank 80H, Pick The
                                        ;Highest Bank Available For Z8 MCU Used
;
;The minimum system initialization code is now complete. For this example,
;system inputs to be debounced are P00-03 and P22-P26.
;
;NOTE: The program assumes all used registers have been previously cleared by
;the power-up initialization routines. This means that inputs start low and
;the debounce counters are set to the low-level threshold. Input levels can be
;set to a user-defined default state (high or low) by initializing the
;debounce counters and I/O port image table registers as required.
;
;
;Set-up To Debounce Port 0:
;
DEBOUNCE: LD      BIT_MASK, #01H      ;Bit Mask Set For Port Bit 0
          LD      MAX_BIT, #10H       ;Max Mask Bit Set For Port Bit 3 (+1)
          LD      PORT, #00H          ;Debounce Port 0
          LD      DB_PTR, #DB_TABLE   ;Initialize The Debounce Cntr Pntr
          LD      INPUT, IM_PORT0     ;Get The Port 0 Image Table Reg
          CALL    SCAN_IN             ;Call The Input Debouncer
          LD      IM_PORT0, INPUT     ;Store Debounced Port 0 Inputs
```

```
;
;Set-up To Debounce Port 2:
;
          LD      BIT_MASK, #04H        ;Bit Mask Set For Port Bit 2
          LD      MAX_BIT, #80H         ;Max Mask Bit Set For Port Bit 6 (+1)
          LD      PORT, #02H            ;Debounce Port 2
          LD      INPUT, IM_PORT2       ;Get The Port 2 Image Table Reg
          CALL    SCAN_IN               ;Call The Input Debouncer
          LD      IM_PORT2, INPUT       ;Store Debounced Port 2 Inputs
          JP      DEBOUNCE              ;Keep Debouncing
;
;*****************************************************************************
;*        SCAN_IN: The following routine debounces the requested input port   *
;*        bits. The port bits must be in sequence to be serviced by this      *
;*        subroutine. The following registers need to be initialized by the   *
;*        user prior to calling the SCAN_IN subroutine:                       *
;*                                                                            *
;*        BIT_MASK = First port bit to debounce (LSB to MSB)                  *
;*        MAX_BIT  = Last port bit to debounce (plus one additional bit pos)  *
;*        PORT     = I/O port number to debounce                              *
;*        DB_PTR   = Reg file ptr to the bit debounce counter table           *
;*        INPUT    = Reg holding debounce port's last state                   *
;*        IM_PORTX = I/O port image register                                  *
;*                                                                            *
;*        In addition, register space must be allocated for the debounce      *
;*        counters — one register per port bit to be debounced. The           *
;*        respective debounce counter is incremented if the bit is high,      *
;*        decremented if the bit is low, until a high- or low-threshold level *
;*        is reached. Transition thresholds are set by the user via the       *
;*        HI_THRESH and LO_THRESH values. When the counter reaches the        *
;*        HI_THRESH value (and the input is still high on the next scan),     *
;*        the port bit is determined to be a valid high level. When the       *
;*        counter reaches the LO_THRESH value (and the input is still low     *
;*        on the next scan), the port bit is determined to be a valid low     *
;*        level.                                                              *
;*****************************************************************************
;
SCAN_IN:  LD      IN_BUFF, @PORT        ;Read The Port Inputs
SCAN_1:   TM      IN_BUFF, BIT_MASK     ;Test For A Logic 0 In The Bit Position
          JR      Z, SCAN_3
          CP      @DB_PTR, #HI_THRESH   ;Test If Debounced Input Is Already Hi
          JR      EQ, SCAN_4
          INC     @DB_PTR               ;Inc This Port Bits Debounce Cntr
```

```
SCAN_2:   INC     DB_PTR                ;Advance Ptr To Next Bit Debounce Cntr
          RL      BIT_MASK              ;Shift Mask For Next Port Bit Position
          CP      BIT_MASK, MAX_BIT     ;Test For Port Bit Debounce Complete
          JR      NE, SCAN_1
          RET                           ;All Required Port Bits Debounced
;
SCAN_3:   CP      @DB_PTR, #LO_THRESH   ;Test If Debounced Input Is Already Lo
          JR      EQ, SCAN_5
          DEC     @DB_PTR               ;Dec This Port Bits Debounce Cntr
          JR      SCAN_2
SCAN_4:   OR      INPUT, BIT_MASK       ;Hi Threshold Valid, Set Port Bit
          JR      SCAN_2
SCAN_5:   COM     BIT_MASK
          AND     INPUT, BIT_MASK       ;Lo Threshold Valid, Reset Port Bit
          COM     BIT_MASK
          JR      SCAN_2
;
          .END
```

APPENDIX B: Z8 CCP MCU SYSTEM DESIGNER'S CHECKLIST

- Ensure that a Z8 MCU decoupling capacitor (0.1uF) is used and located as close as possible to the device's V_{cc} and ground pins.
- Locate the Z8 XTAL/resonator clock source as close as possible to the Z8 XTAL1 and XTAL2 pins.
- Ensure that the Z8 oscillator XTAL/resonator circuit ground trace is routed directly to the Z8 ground pin with no other system connections until at the Z8 ground pin.
- Any system ground trace/plane should be kept as far away as possible from the Z8 oscillator circuit.
- Select the lowest possible Z8 MCU clock speed for the application.
- Any input filtering on Z8 pins should be to the system ground trace as far as possible from the Z8.
- Always use the divide-by-two clock mode for Z8s, especially for those without the clock input glitch filter.
- Ensure that all unused interrupts have their corresponding vector locations defined (example: force a restart).
- Ensure that unused ROM locations are filled with NOP Instructions (with an ending "Jump to Start" instruction).
- Terminate all unused Z8 inputs.
- Protect the XTAL 1 pin from inadvertent Test mode entry by adding a clamping diode to V_{cc}.
- On Z8 OTP devices, protect all EPROM mode pins by adding clamping diodes to V_{cc}. (EPROM modes are activated by voltages > V_{cc} + 1.5V)
- Always use the WDT permanently on ROM mask/OTP programming option.
- Always program WDTMR for WDT on in the HALT and STOP modes.
- Always program all control registers at POR. Do not depend on the MCU reset condition.
- Guardband the use of the HALT, STOP, and WATCHDOG instructions.
- Use fault-tolerant software design techniques whenever possible (as program and register file space allow).

Retro Workshop 果粉工作室

PIC16C54-TO-Z86E02 UPGRADE

THE Z86E02 OFFERS MANY ADVANCED FEATURES COMPARED TO ITS PIC COUNTERPART, AND A PIC16C54-TO-Z86E02 UPGRADE CAN BE DONE—WITHOUT A NEW BOARD LAYOUT!

INTRODUCTION

Customers frequently want to upgrade from a Microchip PIC16C54 microcontroller to a Zilog Z86E02 microcontroller for their application. The reasons for the switch to the Z86E02 include:

- Superior Architectural Features
- Richer Peripheral Set
- More Attractive Price Structure

The PIC16C54-to-Z86E02 upgrade can be accomplished—*without a new board layout.* Although this application note deals specifically with the "how to's" of upgrading from PIC16C54 to Z86E02, our discussion here is not limited to upgrading from PIC16C54 to Z86E02 alone. This application note also applies to upgrading all of Microchip's OTP PIC16C54/56/58 and mask ROM PIC16CR54/58 devices in the 18-pin DIP and SOIC packages to Zilog's Z86X02/04/08 (where X = E, C, or L) microcontrollers in the 18-pin DIP and SOIC packages. The Z86E02/04/08 are OTP versions of the mask ROM Z86C02/04/08 and low-voltage mask ROM Z86L02/04/08 microcontrollers. The final portion of this application note discusses the electrical characteristic considerations in converting PIC OTP and mask ROM devices to Z8 OTP, mask ROM, and low-voltage mask ROM microcontrollers.

MAPPING PIC16C54 PINS TO Z86E02 PINS

The PIC16C54-to-Z86E02 upgrade begins by mapping all of the E02 pins to the PIC. Table 1 shows a pin-for-pin functional best-fit correspondence between the PIC16C54 and the Z86E02.

Table 1. PIC16C54-to-Z86E02 Functional Best-Fit Pin Correspondence

PIC16C54 Pin	Z86E02 Pin	Description
V_{DD}	V_{CC}	Power supply
V_{SS}	GND	Ground reference
OSC1	XTAL1	Crystal oscillator/external clock input
OSC2	XTAL2	Crystal oscillator output
RB0–RB7	P20–P27	Bit-selectable, bi-directional I/Os
RA0–RA1	P00–P01	PIC: Bit-selectable, bi-directional I/Os Z8: Nibble-selectable, bi-directional I/Os
RA2–RA3	P32–P33	PIC: Bit-selectable, bi-directional I/Os Z8: Dedicated input I/Os
T0CKI	P31 (T_{in})	PIC: WDT/Timer0 clock input Z8: Timer/Counter input or dedicated input I/O
/MCLR	P02	PIC: Master clear/reset Z8: E02 has built-in Power On Reset (POR) circuit; hence, an external reset pin is not required. P02 is a nibble-select, bi-directional I/O pin.

Figure 1 shows pictorially how the Z86E02 pins maps to the PIC16C54 pinout per Table 1.

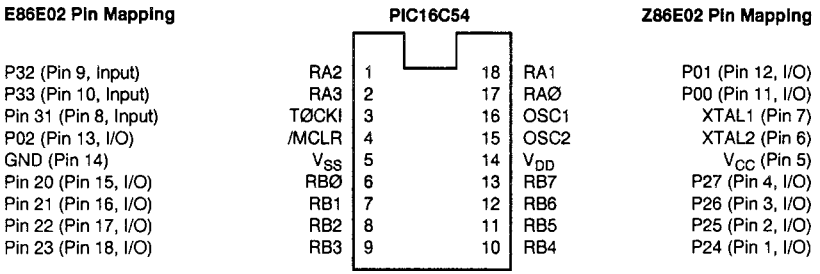


Figure 1. PIC16C54 to Z86E02 Pin Mapping per Table 1

Table 1 demonstrates a direct one-to-one correspondence for PIC16C54 functions V_{DD}, V_{SS}, OSC1, OSC2, RB0-RB7, and T0CK1 to Z86E02 signals V_{CC}, GND, XTAL1, XTAL2, P20-P27, and P31 (T_{in}) respectively. However, an exact functional correspondence for PIC16C54 signals /MCLR and RA0-RA3 does not exist.

As for mapping the PIC /MCLR reset pin to the Z86E02, the E02 has built-in POR circuitry, which obviates the need for an external reset pin on the E02. Thus, the E02 has no corresponding /MCLR pin. When converting from a PIC design, however, you may want to retain the use of an external reset signal. In that case, the E02's P02 may be programmed as an input and periodically polled for a Low state, which indicates a reset condition. When an external reset is detected, the E02 can branch to location 000CH, the program start location, and execute its initialization code as if a POR has occurred.

As for mapping the four PIC RA0-RA3 I/O pins to the Z86E02, you should first attempt to map them to the E02's P00, P01, P32, and P33 I/O pins. P32 and P33 are dedicated input pins and should be mapped to two RA0-RA3 pins that were configured as input I/Os when possible. P00 and P01 are nibble-selectable I/Os, which in conjunction with P02, must be all configured as input or output pins. P00-P02 are not individually configurable as inputs or outputs. Depending on the need, P00 and P01 will be configured as both outputs or both inputs.

If by necessity P00 and P01 are configured as outputs, then P02 has also become an output. Therefore, P02 cannot be used as a polled input for the /MCLR function as previously discussed. In this case, P02 can be used elsewhere as an output I/O and another input I/O can be used for /MCLR. Under less ideal conditions, more trade-offs may be necessary, which include swapping with some of the E02's P20-P27 I/O pins to satisfy the input and output I/O requirement for this conversion.

UPGRADING WITHOUT ADDITIONAL LAYOUT

Under ideal conditions, upgrading to an E02 from a PIC16C54 can be performed without requiring a new board layout. In the case that the application does not require the counter for external event counting and RA0-RA3 of the PIC were used as inputs, the conversion is done by simply rotating the Z86E02 by 180° to fit into a PIC16C54 socket. See Table 2 for the PIC16C54-to-Z86E02 pin correspondence for this ideal fit.

Table 2. PIC16C54 to Z86E02 Ideal Mapping

PIC16C54 Pin	Z86E02 Pin	Description
V_{DD}	V_{CC}	Power supply
V_{SS}	GND	Ground reference
OSC1	XTAL1	Crystal oscillator/external clock input
OSC2	XTAL2	Crystal oscillator output
RB0 - RB7	P20 - P27	Bit-selectable, bi-directional I/Os
RA0 - RA2	P31 - P33	PIC: Bit-selectable, bi-directional I/Os configured as inputs Z8: Dedicated input I/Os
RA3	P00	PIC: Bit-selectable, bi-directional I/O configured as an input Z8: Nibble-selectable, bi-directional I/O configured as an input
T0CKI	P01	PIC: WDT/Timer0 clock input (not used) Z8: Nibble-selectable, bi-directional I/O configured as an input
/MCLR	P02	PIC: Master clear/reset Z8: Nibble-selectable, bi-directional I/O configured as an input

Figure 2 shows pictorially how the Z86E02 pins map to the PIC16C54 pinout under the ideal situation per Table 2. You can see that the E02 can indeed be rotated upside-down 180° to fit in the PIC16C54 DIP or SOIC socket.

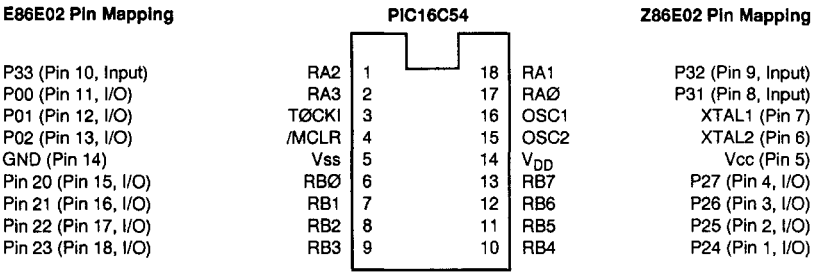


Figure 2. PIC16C54 to Z86E02 Ideal Pin Mapping per Table 2

ELECTRICAL CHARACTERISTIC CONSIDERATIONS

When migrating to the Z86E02 from PIC16C54, you must carefully consider some electrical characteristic differences between the devices. One consideration is the difference in the operational supply voltage ranges of the two devices. The second consideration is the difference in the buffer type of the two devices.

Operational Voltage Range Differences

The PIC family of devices and Z8 family of devices have very different supply voltage operation ranges. These differences are important design considerations in upgrading to the Z8 from PIC, especially in hand-held type of applications where batteries are used. See Table 3 for a listing of Microchip's PIC16C54 types of devices and their supply voltage (V_{DD}) ranges. The PIC16C54 and the low-voltage PIC16LC54 with all their varying oscillator types are OTP microcontrollers, while the PIC16CR54s are mask ROM devices. Microchip's PIC supply voltage ranges vary both by device type and oscillator option.

Table 3. PIC16C54 Devices Supply Voltage Ranges

OSC. Option	PIC16C54 V_{DD} Range	PIC16LC5 V_{DD} Range	PIC16CR54 V_{DD} Range
RC	3.0V–6.25V	2.5V–6.25V	2.0V–6.25V
LP	3.0V–6.25V	2.5V–6.25V	2.0V–6.25V
XT	3.0V–6.25V	2.5V–6.25V	2.0V–6.25V
HS	4.5V–5.5V	N/A	4.5V–5.5V

Contrast and compare Table 3 with Table 4 where the Z86E02 OTP, mask ROM Z86C02, and low-voltage mask ROM Z86L02 supply voltage (V_{CC}) ranges are tabulated.

Table 4. Z86E02/C02/L02 Supply Voltage Ranges

Z86E02 V_{CC} Range	Z86C02 V_{CC} Range	Z86L02 V_{CC} Range
4.5V–5.5V	3.0V–5.5V	2.0V–3.9V

The designer must ensure that the target Z8 device of the PIC-to-Z8 conversion has an acceptable operating voltage range for the application. (For the sake of preserving each manufacturer's nomenclature, we will continue to use V_{DD} to refer to the operational supply voltage range for a PIC device and V_{CC} to refer to the operational supply voltage range for a Z8 device.)

Buffer Type Differences

Another difference between PIC and Z8 microcontrollers is the difference in buffer types. The PIC has a TTL buffer type, while the Z8 has a CMOS buffer type. If a Z86E02 were to replace a PIC16C54, the buffer type of the surrounding logic may have to be changed to compatible CMOS logic from TTL logic.

Examine, compare, and contrast some of the electrical characteristics to see why it may be necessary to change the buffer types of the surrounding logic. See Table 5, which lists the Input Low-Voltage Maximum (V_{IL} Max) and Input High-Voltage Minimum (V_{IH} Min) for the PIC16C54 device types and Z86X02 device types. In the case of the PIC devices, the electrical characteristic values for the I/O ports are used, whereas, the Z8 electrical characteristic values apply to any and all input pins.

Table 5. PIC versus Z8 Input Voltage Requirements

PIC Device	V_{IL} Max	V_{IH} Min	Z8 Device	V_{IL} Max	V_{IH} Min
PIC16C54	0.2 V_{CC}	0.45 V_{CC}	Z86E02	0.2 V_{CC}	0.7 V_{CC}
PIC16LC54	0.2 V_{CC}	0.45 V_{CC}	Z86C02	0.2 V_{CC}	0.7 V_{CC}
PIC16CR54	0.2 V_{CC}	0.45 V_{CC}	Z86L02	0.1 V_{CC}	0.9 V_{CC}

It appears that the V_{IL} Max values for both Z8 and PIC are the same except for the Z86L02. The main difference between PIC and Z8 is the V_{IH} Min value. The nominal difference is 25 percent of V_{CC} and in the extreme case of the Z86L02 versus PIC, it is 45 percent of V_{CC}. The minimum input voltage level required for a logic "1" High is much greater with the Zilog Z86X02s. Next, examine, compare, and contrast the Output Low-Voltage Maximum (V_{OL} Max) and the Output High-Voltage Minimum (V_{OH} Min) for PIC and Z8 (see Table 6). As before, the electrical characteristic values for the PIC I/O ports are used, whereas, the Z8 electrical characteristic values apply to any and all output pins.

Table 6. PIC versus Z8 Output Voltage Requirements.

PIC Device	V_{OL} Max	V_{OH} Min	Z8 Device	V_{OL} Max	V_{OH} Min
PIC16C54	0.6V @ 4.5V V_{DD}	V_{DD}–0.7V	Z86E02	0.4V @ 4.5V V_{CC}	V_{CC}–0.4V
PIC16LC54	0.6V @ 4.5V V_{DD}	V_{DD}–0.7V	Z86C02	0.4V @ 5.5V V_{CC}	V_{CC}–0.4V
PIC16CR54	0.5V @ 6.0VV_{DD}	V_{DD}–0.5V	Z86L02	0.4V @ 3.9V V_{CC}	V_{CC}–0.4V

In Table 6, the Z8 V_{OH} values are 0.1V to 0.3V higher than PIC V_{OH} values due to the Z8's CMOS output buffers. Also note that the Z8 V_{OL} values are approximately 0.1V to 0.2V lower than PIC V_{OL} values.

In general, the required Z8 input voltages and output voltages will be "closer" to its positive supply voltage and negative supply/ground voltage than PIC microcontrollers due its CMOS buffers. The required PIC input voltages and output voltages are "farther away" from its positive supply voltage and negative supply/ground voltage than a Z8 microcontroller due to its TTL buffers. Again, if a Z86E02 were to replace a PIC16C54, the buffer type of the surrounding logic may have to be changed to compatible CMOS logic from previous TTL-type logic. In some cases, choice placement of pull-up resistors maybe all that is required, while in other cases the type of logic surrounding the microcontroller will have to be completely changed.

SUMMARY

If you choose to upgrade from a PIC16C54 to a Z86E02, first carefully map PIC16C54 pins to Z86E02 pins. Under the right circumstances, the Z86E02 could be rotated 180° relative to the PIC16C54 chip orientation and placed right into the PIC socket. Accomplishing the 180° rotation will mean that a new board layout will not be required to upgrade. You should also consider the two main electrical characteristic differences:

- Operational Supply Voltage V_{DD}/V_{CC} Ranges
- TTL/CMOS Buffer Types

Understanding the difference in operational voltage ranges will help you pick the best Z86X02 target device for the application. Understanding the difference in buffer type between a PIC and a Z8 will ensure that the application will function electrically after the upgrade is performed.

USING A Z8 MCU FOR POWER PHASE CONTROL

ZILOG Z8® MCUS ARE EXCELLENT DEVICES FOR ALL APPLIANCE APPLICATIONS, AND THE COMPARATOR FEATURE MAKES THEM ESPECIALLY WELL-SUITED FOR APPLICATIONS USING POWER PHASE CONTROL.

INTRODUCTION

There are many reasons for using MCUs to control electrical appliances instead of older, conventional power controllers. For starters, MCUs regulate power precisely and efficiently.

- MCUs can turn on the firing angle accurately, and the firing angle can be controlled intelligently by the MCU without the component value changing.
- In the world of digital communication, MCUs can receive the command from the central commander. Therefore, home/office central control systems can be implemented to control heating or lighting. Most of the electrical appliances can be controlled by this technique. MCUs even help to link up all the appliances to the central control system.
- MCUs can control an electrical appliance with varying power outputs at different timing and sequential patterns. Rhythmical electric fans use such a technique to simulate the natural wind. The different response of various servos also can be adjusted with the intelligence of the MCU design.
- MCUs can be used in other control functions such as infrared remote control, key detection, and LED indication. Intelligent electrical appliances can be more competitively priced than less intelligent appliances.
- MCUs can be used with digital or analog sensors to measure the temperature with a non-linear thermistor, used in temperature or fan control.

Built-In Comparators—A Distinctive Zilog Z8 MCU Advantage

Zilog's Z8® MCUs have built-in comparators—a special feature useful in detecting the zero crossing of the AC sine wave. The comparators can detect 100V / 200V for auto-range switching, and also can be used as an analog-to-digital converter (ADC) to interface with the analog sensors. This application note explains how MCUs, in general, and Zilog's Z8 MCUs, in particular, are well-suited for power phase control. Zilog's Z8 MCUs are excellent devices for all appliance application because Zilog Z8s—from cost-effective to advanced devices—have the important comparator feature.

THE BASICS OF POWER PHASE CONTROL

In AC power system applications, the amount of power delivered to the system load can be varied by controlling the phase of the input voltage or current waveforms. This method is commonly used in the industry to control motors, electric heaters, light dimmers, and so on. The power delivered to the load is equal to the product of the voltage times the current:

$P = V \times I$

Power line waveforms are 50/60 Hz AC sine waves. The applied power can be controlled by controlling the amount of time per cycle in which voltage and current is applied to the system load. Thus, if you have an open circuit condition, the current/voltage and hence the power delivered will be zero. Triacs are devices that can be used to switch on and off the power to a load in AC power applications.

The power delivered to load is equal to the integration of V x I from turn-on fire angle ϕ to turn-off phase angle π.

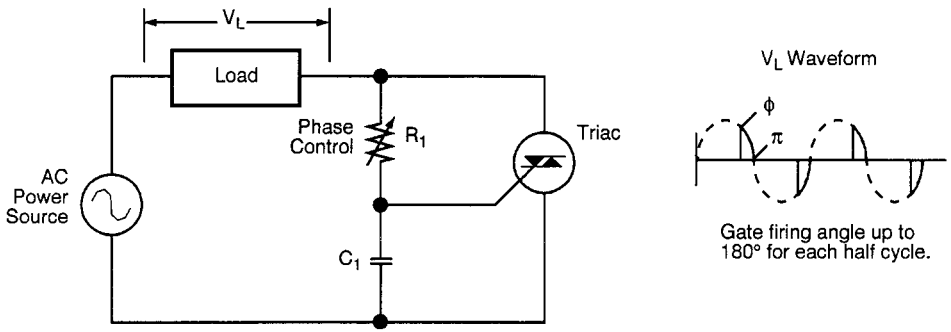


Figure 1. Fire Angle Control

A gate voltage turns on the triac. Once turned on, the triac will generate a substantial current to maintain the gate voltage until the AC sine wave approaches zero voltage.

Figure 1 shows a simple RC phase-delay gate circuit. By varying R1, the turn-on fire angle ϕ can be controlled.

This simple trigger circuit is seldom used in industry due to the slow-rising gate current and voltage level. Required gate-trigger signal levels vary due to differences between triacs, the quadrant of operation, and temperature. Such limitations make it difficult to calibrate phase-control potentiometer R1 for accurate firing angle. Most designers will add a trigger device such as a diac or a triac driver to improve the performance (see Figure 2). If isolation between the AC line and the control circuit is required, an opto-triac driver can also be used.

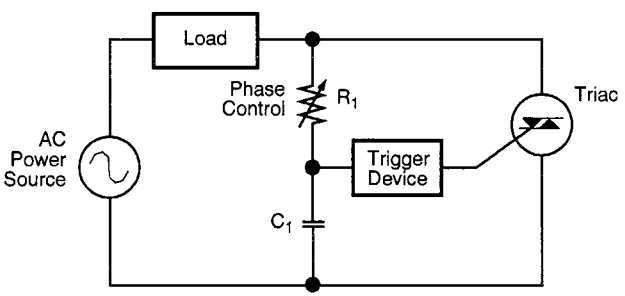


Figure 2. Basic Triac Phase-Delay Gate Circuit with Trigger

USING AN MCU—THEORY OF OPERATION

Zero-Crossing Detection

The power line waveform is an AC sine wave with a frequency of 50/60 Hz (see Figure 3). The AC line is connected in series to a 1M Ohm resistor that limits the current to the MCU. The two diodes clamp the voltage within V_{CC} + 0.7V and -0.7V, resulting in a rectangular wave to an MCU.

50/60 Hz Frequency Counting

The MCU counts the period of the AC sine wave in order to detect whether the line frequency is 50 Hz or 60 Hz. An internal timer is used to count the period of signal.

For a 50 Hz power line, the cycle period is equal to 1/50 = 20 milliseconds (ms). There are two zero crossings in each cycle: one at $\theta = 0$, the other at $\theta = \pi$. The time between these two crossing is 10 ms for 50 Hz [(1/50) Hz / 2] and 8.33 ms for 60 Hz [(1/60) / 2]. The power line frequency can be determined by counting the time between these two zero crossings.

110V/220V Line Voltage Detection

Line voltage can also be determined by using a potential divider (see Figure 4). After the MCU detects the zero-crossing point, it can timeout after a $\pi / 2$ period, then do an A/D conversion with the comparator. The value from a 220V AC will be much greater than that from the 110V input. At $\pi / 2$, the 220V AC gives a peak voltage of 220V x square root 2 = 311V. After the potential divider of 101 to 1, it becomes 3.08V. Similarly, the 110V AC will produce a 1.54V after the potential divider. With a simple A/D conversion, the AC line voltage can be determined.

Firing Angle and Power Delivery

As previously discussed, the power delivery can be controlled by the firing angle. Power delivered is equal to the integration of the V x I result from fire angle ϕ to π. The triac will conduct the power after the firing angle and turn off power at the zero-crossing point.

P = Integration of (V^2 / R) from ϕ to π.

$V = V_o \sin \theta$.

P = Integration of $(V_o^2 \sin^2 \theta / R)$ from ϕ to π.

= (V_o^2 / R) x Integration of $(\sin^2 \theta)$ from ϕ to π.

= (V_o^2 / R) x Integration of $(1/2)(1 - \cos 2\theta)$ from ϕ to π.

= $(V_o^2 / R) \times \{\pi / 2 - \phi / 2 + [\sin (2\phi)] / 4\}$.

Full power is when $\phi = 0$, power = $(V_o^2 / R) \times \pi / 2$. Zero power is when $\phi = \pi$.

The ratio of power delivery to the full power is $\{\pi / 2 - \phi / 2 + [\sin (2\phi)] / 4\} / (\pi / 2)$.

From the power ratio equation, changes in the firing angle can determine differences in power ratio control (see Table 1).

Table 1. Power Delivery Ratio Example

Power Ratio	Angle ϕ
1.0000	0.0000
0.9001	0.8130
0.8000	1.0565
0.7002	1.2450
0.6002	1.4120
0.5002	1.5705
0.4001	1.7290
0.3001	1.8960
0.2000	2.0850
0.1001	2.3280
0.0000	3.1416

THEORY OF HARDWARE IMPLEMENTATION

Zero-Crossing Detection TTL Level (Alternative)

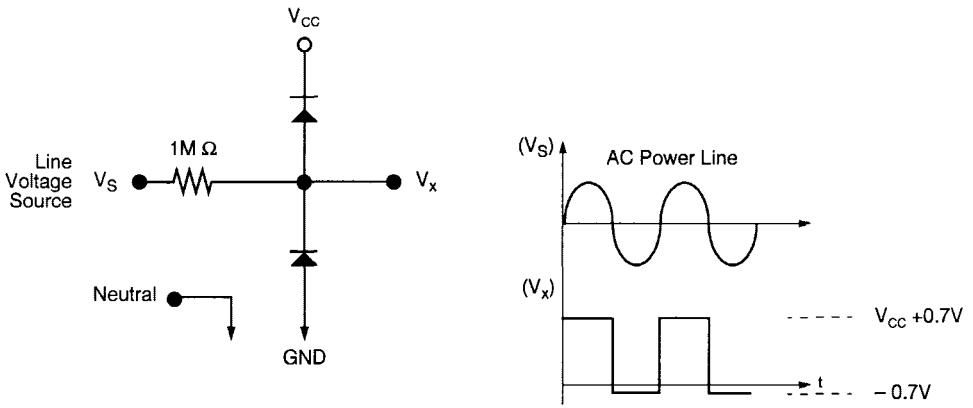


Figure 3. Zero-Crossing Detection

110V/220V AC Line Voltage Discrimination

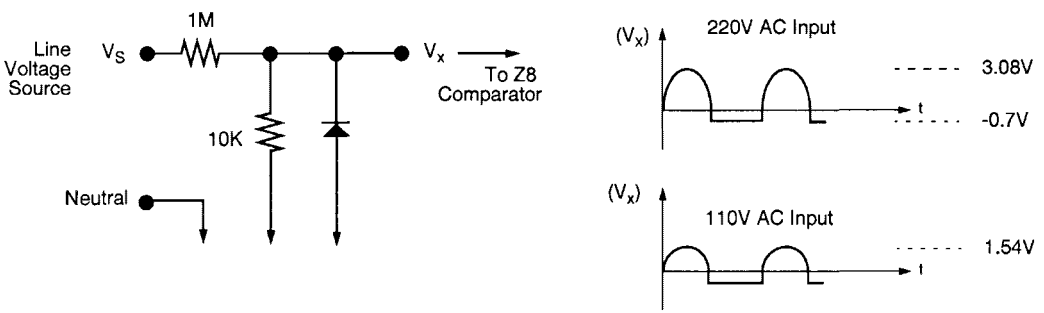


Figure 4. 120V/220V AC Line Voltage Discrimination

Triac Switching

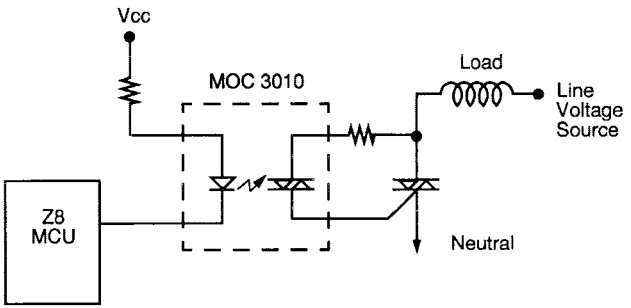


Figure 5. Opto-Isolated Triac Connection

EXCEEDING MAXIMUM SPECIFICATIONS

The voltage input to the Z8® MCU comparator in the example hardware implementations (see Figure 3 and Figure 4) exceeds the absolute maximum specification of 1) Voltage Input Low on any pin = V_{SS} - 0.6V and 2) Voltage Input High on any pin = V_{CC} + 0.6V.

Exceeding these two maximum specifications is permissible as long as the input current to the Z8 MCU does not exceed ±600 µA.

SCHEMATIC

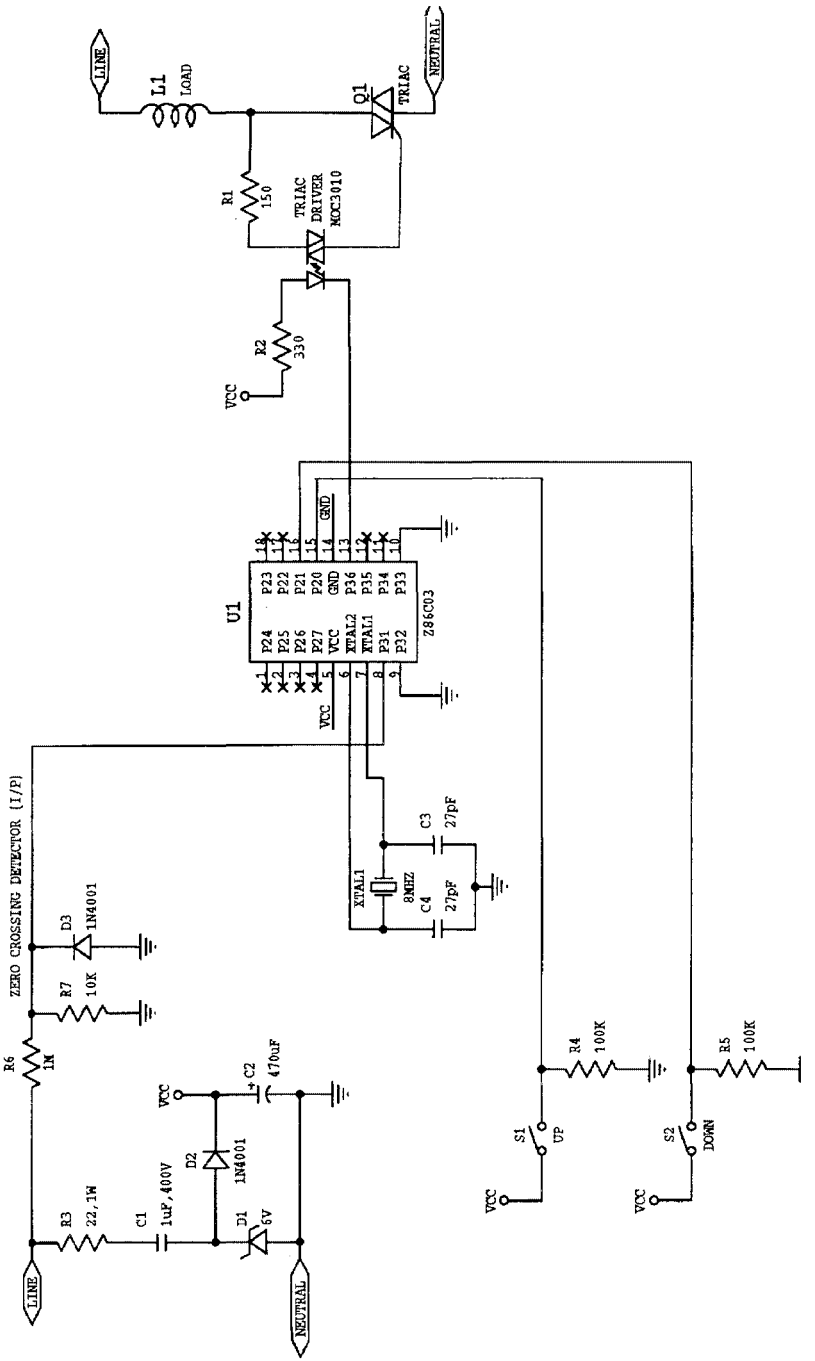


Figure 6. Power Line Phase Control Schematic

SOFTWARE IMPLEMENTATION

50/60 Hz Detection

When the device is powered up, the first few power cycles may not be stable. The timing and amplitude of the comparator input voltage will depend on the loading and capacitance. Therefore, there is a delay for 40 ms in order to neglect the first few cycles in power up. After the first valid zero crossing of the AC line, the Timer 1 is initialized with an initial value of 250, prescaler of 50.

Oscillation crystal	:	8 MHz
Clock to the timer 1	:	8 MHz / 8 = 1 MHz
Prescaler 1	:	50
Initial value	:	250
Timer 1 will expire	:	[1 / (1 MHz)] x 50 x 250 = 12.5 ms

The initial value of the timer will keep on decrementing for every 50 microsecond (µs). If the AC line is 50 Hz, the next zero crossing will be at (1/50 Hz) /2 = 10 ms. If Timer 1 is read again, it should show 10 ms / 50µs = 200. If the AC line is 60 Hz, the Timer 1 value should be 8.3 ms / 50µs = 167. By comparing the Timer 1 value, the correct frequency of the AC line can be determined. The detail of the AC frequency detection can be found in the "check_Hz" subroutine.

Firing Angle for 50/60 Hz AC

The triac is turned on for 100 µs at the firing angle. 50 Hz firing angle is different from the 60 Hz firing angle. However, we can accompany both by setting different prescaler value in the Timer 0. After we detect the 50 Hz / 60 Hz AC line at power up, we store 50 in the prescaler 0 if the AC frequency is 60 Hz and 60 for the 50 Hz frequency. With 8 MHz crystal oscillation, the internal clock period will be 1 µs.

Table 2. Firing Angle for 50/60 Hz AC

		Firing Time (S) = 10 ms * (ϕ / π)	Firing Time (S) = 8.3 ms * (ϕ / π)	T1 Value = Firing Time / (60 * 1 µs)	T1 Value = Firing Time / (50 * 1 µs)
Firing Angle ϕ	**Power Ratio**	**50 Hz AC**	**60 Hz AC**	**50 Hz AC**	**60 Hz AC**
0.0000	1.0000	0.0000	0.0000	0	0
0.8130	0.9001	0.0026	0.0022	43	43
1.0565	0.8000	0.0034	0.0028	56	56
1.2450	0.7002	0.0040	0.0033	66	66
1.4120	0.6002	0.0045	0.0037	75	75
1.5705	0.5002	0.0050	0.0042	83	83
1.7290	0.4001	0.0055	0.0046	92	92
1.8960	0.3001	0.0060	0.0050	101	101
2.0850	0.2000	0.0066	0.0055	111	111
2.3280	0.1001	0.0074	0.0062	124	124
3.1416	0.0000	0.0100	0.0083	167	167

After the zero crossing, the T0 value is loaded into the Timer 0 initial value. With the initial value count down to zero, Timer 0 interrupt will be served. At this firing angle, the MCU will turn on the triac for 100 µs. After the triac is on, it will generate substantial current to keep the triac on until the AC power line move to the zero crossing again. Therefore, different firing angle can control different ratio of power delivery.

Key Scanning

The power level can be changed by an increment button and a decrement button. These two buttons can used to select the power level from 0 to 10. Key scanning is performed once in 9 AC cycles (180 ms for 50 Hz frequency or 150 ms for 60 Hz frequency).

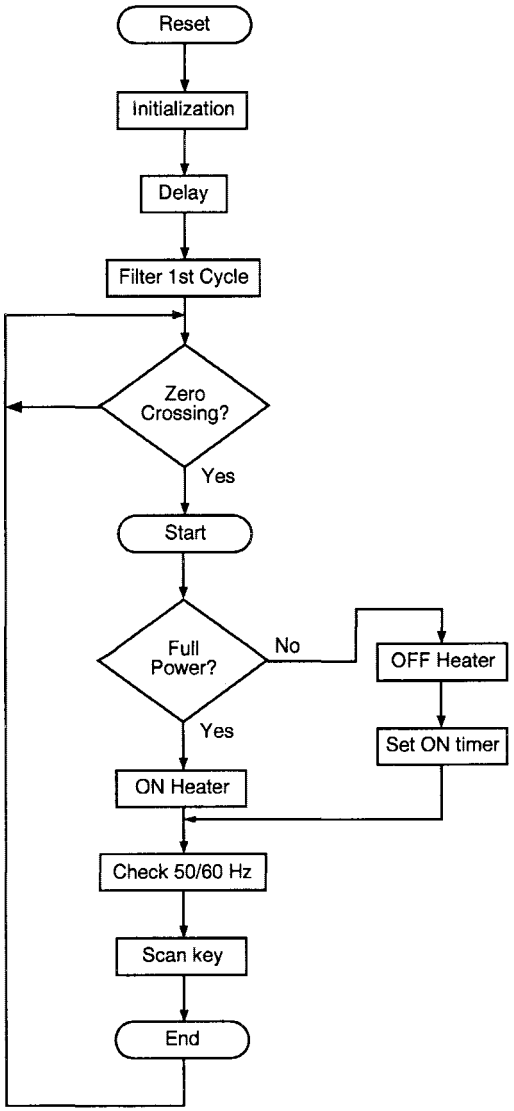


Figure 7. Software Flowchart

SOFTWARE CODE EXAMPLE

```
;*********************************************************************
;       FILE    :PHASE.S
;       MCU     :Z86C03
;       PROJECT :PHASE CONTROL POWER APPLIANCE
;       SOFTWARE:REVISION 1.1
;
;       THIS PROGRAM IS ASSEMBLED BY THE ZILOG ZASM ASSEMBLER.
;*********************************************************************
bitno0          .equ    %01
bitno1          .equ    %02
bitno2          .equ    %04
bitno3          .equ    %08
bitno4          .equ    %10
bitno5          .equ    %20
bitno6          .equ    %40
bitno7          .equ    %80

;*********************************************************************
;               PORTS DEFINITION
;*********************************************************************
;       Port 2 pin
;       P22-P27 output idle N.C.
.upkey          .equ    bitno0          ; temp. UP key
.downkey        .equ    bitno1          ; temp. DOWN key

;       Port 3 pin
;       P31 : AC zero-crossing detection
;       P33 : voltage reference
;       P32 : idle input, pull to ground
;       P34,35 : idle output N.C.
.acinput        .equ    bitno1
.reference      .equ    bitno3          ; ground
.SWITCH         .equ    bitno6          ; ON/OFF switch
                                        ; 0 = ON ; 1 = OFF

;*********************************************************************
;               REGISTERS DEFINITION
;*********************************************************************
status          .equ    r0
temp_level      .equ    r1
temp            .equ    r2
pwrout          .equ    r3
t_chkkey        .equ    r4              ; 180 ms count for scan key
tempirq         .equ    r6
phasetable      .equ    %20

;*********************************************************************
;               BITS DEFINITION
;*********************************************************************
;       status register
;-----------------------------------------
.chkHz          .equ    bitno1
.firstcount     .equ    bitno2
```

```
;*******************************************************************
;               CONSTANTS DEFINITION
;*******************************************************************
tc_chkkey       .equ    18              ; 180 ms timer
k_fifty         .equ    11110000b       ; pres=60 single pass
k_sixty         .equ    11001000b       ; pres=50 single pass
hi_hz           .equ    100             ; hi_hz, mid_hz, low_hz for
mid_hz          .equ    67              ; checking 50/60 Hz frequency
low_hz          .equ    30
start_level     .equ    5               ; power start level =5

;*******************************************************************
;               MACROS
;               REFER TO Z8 TECHNICAL MANUAL FOR MACRO DEFINITION.
;*******************************************************************
bset:       .macro  arg1,arg2
       or       arg1,#arg2
       .endm

bclr:       .macro  arg3,arg4
       and      arg3,#^c (arg4) ^& 0ffh
       .endm

brset:      .macro  arg1,arg2,arg3
       tm       arg1,#arg2
       jr       nz,arg3
       .endm


brclr:      .macro  arg1,arg2,arg3
       tm       arg1,#arg2
       jr       z,arg3
       .endm

SWITCHON:   .macro
       bclr     p3,.SWITCH
       .endm

SWITCHOFF:  .macro
       bset     p3,.SWITCH
       .endm

;*******************************************************************
;               INTERRUPTS VECTOR
;*******************************************************************

       .mlist   on
       .list    on
; Interrupt vector address %00 to %0C
       .ORG     %0000
       .word    dummy
       .word    dummy
       .word    dummy
       .word    dummy
       .word    timer0
       .word    dummy
```

```
;***********************************************
;              PROGRAM STARTS HERE              *
;***********************************************
BEGINNING:  di
        ld      P01M,#00000100b         ; P0, P1 output,
                                        ; internal stack
        ld      P2M,#00000011b          ; P20,P21 input
                                        ; P22-P27 output
        ld      p2,#%ff
        ld      P3M,#%02                ; P3 analog
                                        ; P2 open-drain
        ld      P3,#%ff                 ; set P3 = 1,
                                        ; P36 turn off the triac
        clr     SPH
        ld      SPL,#126                ; INIT STACK POINTER
        ld      IPR,#00010011b
        ld      IMR,#0                  ; disable all interrupts

; INITIALIZE RAM TO "0"
        srp     #%40
        ld      R14,#%3f
ZRAM:       clr     @R14
        djnz    R14,ZRAM

        SWITCHOFF                       ; switch off power

;       Initialize all the registers

        srp     #%10                    ; set working register to %10
        ld      t_chkkey,#tc_chkkey
        ld      phasetable,#255         ; zero power
        ld      phasetable+1,#124       ; 1/10 power
        ld      phasetable+2,#111       ; 2/10 power
        ld      phasetable+3,#101       ; 3/10 power
        ld      phasetable+4,#92        ; 4/10 power
        ld      phasetable+5,#83        ; 5/10 power
        ld      phasetable+6,#75        ; 6/10 power
        ld      phasetable+7,#66        ; 7/10 power
        ld      phasetable+8,#56        ; 8/10 power
        ld      phasetable+9,#43        ; 9/10 power
        ld      phasetable+10,#0        ; full power
        ld      pwrout,#start_level     ; preset power level
        bset    status,.chkHz           ; first-time frequency check
        bset    status,.firstcount
        call    delay40ms               ; wait for AC to be stable
wait:
        brclr   p3,.acinput,wait        ; filter first signal
;*********************************************************************************************
;/////////////////////////////////////////////////////////////////////////////////////////////
;       Wait for AC zero-crossing input.
;       Zero crossing can be negative-to-positive and positive-to-negative.
;       ACcrossing subroutine is only called when AC line changes its polarity (twice per cycle).
;/////////////////////////////////////////////////////////////////////////////////////////////
Main:
        brset   p3,.acinput,Main
        call    ACcrossing
waitone:
        brclr   p3,.acinput,waitone
        call    ACcrossing
        jr      Main
```

```
;*********************************************************************

;-----------------------------------------------------
;               SUBROUTINES
;-----------------------------------------------------

;/////////////////////////////////////////////////////
;       AC line change in every 10 ms for 50 Hz.
;/////////////////////////////////////////////////////
ACcrossing:
       ld      temp,phasetable(pwrout)
       and     temp,temp
       jr      nz,phase_control
fullpoweron:                            ; turn on power
       SWITCHON
       jr      end_phase

phase_control:
       SWITCHOFF                        ; turn off power
       ld      t0,temp
       ld      tmr,#00000011b           ; enable T0, reload timer
end_phase:
       call    check_Hz                 ; check 50/60 Hz
       call    scankey                  ; scan keys
ret_AC:
       ret

;/////////////////////////////////////////////////////
;               Check 50/60 Hz frequency
;/////////////////////////////////////////////////////
check_Hz:
       brclr   status,.chkHz,end_check_hz
; Check 50/60 Hz for 1st AC cycle.
       brclr   status,.firstcount,secondcount
       ld      t1,#250
       ld      pre1,#11001010b          ; pre1=50, single pass
       ld      tmr,#00001100b           ; enable T1
       clr     irq
       ei                               ; enable interrupt
       bclr    status,.firstcount
       jr      end_check_hz
secondcount:
       ld      temp,t1
       cp      temp,#low_hz
       jr      c,out_Hz
       cp      temp,#mid_hz
       jr      c,fifty_hz
       cp      temp,#hi_hz
       jr      c,sixty_hz
out_Hz:
       ld      tmr,#00001100b           ; enable T1 / count again
       jr      end_check_hz
sixty_hz:
       ld      pre0,#k_sixty            ; prescaler = 50
       jr      end_count_hz
fifty_hz:
       ld      pre0,#k_fifty            ; prescaler = 60
end_count_hz:
       bclr    status,.chkHz
       ld      tmr,#00000011b           ; enable T0, disable T1
       clr     irq
       ld      imr,#10010000b           ; enable T0, disable T1
end_check_hz:
```

```
        ret

;delay40ms
        call    delay10ms
        call    delay10ms
        call    delay10ms
        call    delay10ms
        ret

;////////////////////////////////////////////////////
;       1.5 µS x  (30x222)  =  10 ms
;////////////////////////////////////////////////////
delay10ms:
        ld      temp,#222               ; 6 cycles
loop1:
        nop                             ; 6 cycles
        nop                             ; 6 cycles
        dec     temp                    ; 6 cycles
        jr      nz,loop1                ; 12 cycles
        ret

;////////////////////////////////////////////////////
;       1.5 µS x  (18x4)  =  108 µS
;////////////////////////////////////////////////////
delay100µS:
        ld      tempirq,#2              ; 10 cycles
loop2:
        dec     tempirq                 ; 6 cycles
        jr      nz,loop2                ; 12 cycles
        ret

;////////////////////////////////////////////////////
;       Scan key in every 9 AC cycles
;       That is, 20 ms x 9   = 180 ms for 50Hz AC
;       16.7 ms x 9 = 150 ms for 60Hz AC
;////////////////////////////////////////////////////
scankey:
        dec     t_chkkey
        jr      nz,end_scankey
        ld      t_chkkey,#tc_chkkey     ; reset 180 ms counter
        brset   P2,.upkey,incpreset     ; check increment key
        brset   P2,.downkey,decpreset   ; check decrement key
        jr      end_scankey
```

```
;////////////////////////////////////////////////////////
;       Increment the PWM output power if UP key is pressed.
:       Maximum PWM level = 10
;////////////////////////////////////////////////////////

incpreset:
        brset   P2,.upkey,incpreset
        inc     pwrout
        cp      pwrout,#10
        jr      le,end_scankey
        ld      pwrout,#10
        jr      end_scankey

;////////////////////////////////////////////////////////
;       Decrement the PWM output power if DOWN key is pressed.
;       Minimum PWM level = 0
;////////////////////////////////////////////////////////

decpreset:
        brset   P2,.downkey,decpreset
        dec     pwrout
        jr      pl,end_scankey
        ld      pwrout,#0
end_scankey:
        ret

;////////////////////////////////////////////////////////////////////////////
;       Timer interrupt to turn on the triac for 100 µs.
;       Triac will generate a substantial current to keep it on after 100 µs ON.
;////////////////////////////////////////////////////////////////////////////

timer0:
        SWITCHON                        ; Turn ON triac opto-driver
        call    delay100uS              ; 100 µS turn on time
        SWITCHOFF                       ; Turn OFF triac opto-driver
        iret
;dummy
        iret

        .end
```

ANALOG-TO-DIGITAL CONVERSION TECHNIQUES USING ZILOG Z8® MCUS

A SEPARATE ADC CHIP ISN'T ALWAYS NECESSARY. MANY APPLICATIONS REQUIRING A/D CONVERSIONS CAN BE ACHIEVED WITH 8-BIT MCUS—AND WITHOUT COMPROMISING ACCURACY, SPEED, OR SYSTEM COST.

INTRODUCTION

Many embedded controller applications require an analog voltage to be captured. Depending on the application, a separate A/D converter (ADC) chip may be necessary because of the speed and resolution requirements. However, many designs do not need fast conversion speeds, and 8 bits of resolution is adequate. For instance, a digital thermostat samples the temperature periodically and turns the heater or air conditioner on or off when the temperature hits a trip point. Here, the measurement speed for the voltage across a thermistor is not critical since the temperature is changing rather slowly. In this case, conversion times on the order of milliseconds are acceptable.

Capturing fast-changing signals, such as audio, requires a much faster conversion rate. If the highest audio frequency is 4 kHz coming into the ADC, the sample rate should be at least twice this (8 kHz). Because of the limited processing time in between samples (in this case 125 µs), the ADC must do a conversion quickly, so the MCU has time to process the data before the next sample.

Since most designs are cost-sensitive—especially in consumer electronics—there may not be the luxury of adding relatively expensive ADC chips to the design. Design engineers must look for a more integrated solution, and Zilog has the solution.

ZILOG Z8 MCU FEATURES

Zilog Z8® microcontrollers (MCUs) to which the following A/D conversion techniques apply range in package sizes from 18 to 44 pins. ROM sizes vary from 0.5 to 4 KB. These are available in masked and OTP configurations, and can be clocked up to 12 MHz using a RC, LC, ceramic resonator, or crystal oscillator (refer to Table 1).

Two power-down modes are available: HALT and STOP.

- HALT freezes code execution, but leaves the timers enabled. The processor can exit HALT via an external or internal interrupt.
- STOP completely shuts down the chip, including the oscillator. To come out of STOP mode, you apply a positive- or negative-going edge to one of the external digital inputs or via the internal Watch-Dog Timer (WDT). This application is set up via the Stop-Mode Recovery (SMR) register.

The block diagram of a typical Z8 architecture is shown in Figure 1. Selected Zilog MCUs are shown in Table 1.

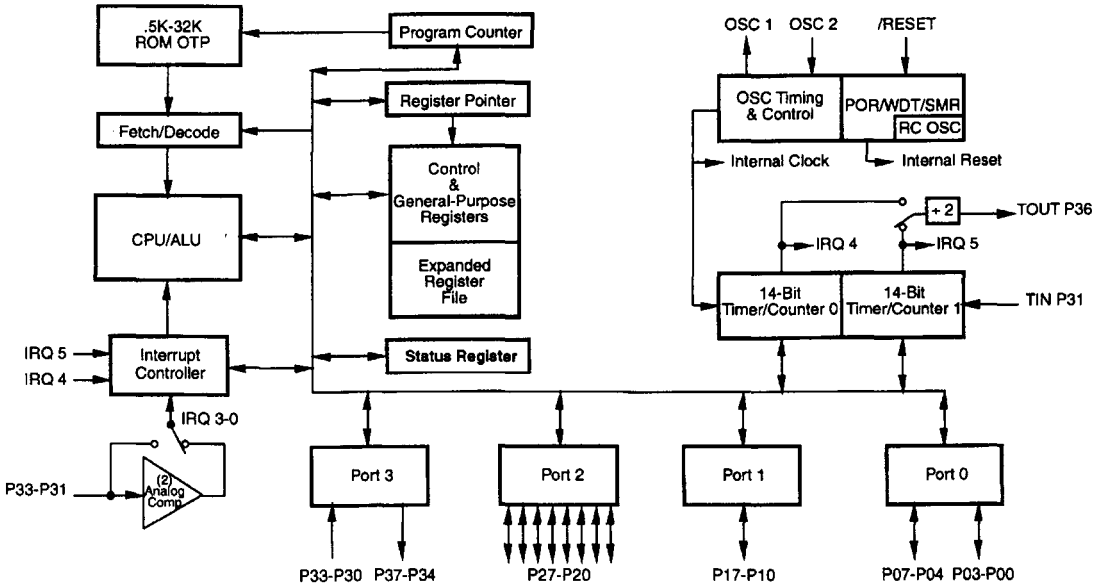


Figure 1. Typical Z8 MCU Block Diagram

Table 1. Selected List of Zilog Z8 MCUs

Product Family	Pin Count	ROM (KB)	RAM (Bytes)	I/O	Speed (MHz)	Package Type
Z86C03	18	0.5	64	14	8	DIP, SOIC
Z86E03	18	0.5	64	14	8	DIP, SOIC
Z86C04	18	1	128	14	8	DIP, SOIC
Z86E04	18	1	128	14	8	DIP, SOIC
Z86C06	18	1	128	14	4, 12	DIP, SOIC
Z86E06	18	1	128	14	4, 12	DIP, SOIC
Z86C07	18	2	128	14	8, 12	DIP, SOIC
Z86E07	18	2	128	14	8, 12	DIP, SOIC
Z86C08	18	2	128	14	4, 12	DIP, SOIC
Z86E08	18	2	128	14	4, 12	DIP, SOIC
Z86C09	18	2	125	14	12	DIP, SOIC
Z86C16	18	1	128	14	8	DIP, SOIC
Z86C19	18	4	125	14	12	DIP, SOIC
Z86C30	28	4	256	24	12	DIP, SOIC
Z86E30	28	4	256	24	12	DIP, WIN
Z86C31	28	2	128	24	8	DIP
Z86E31	28	2	128	24	8	DIP
Z86C32	28	2	237	24	4, 8	DIP, SOIC
Z86C40	40/44	4	256	32	4, 12	DIP, PLCC, QFP
Z86E40	40/44	4	256	32	12	DIP, PLCC, QFP

Note: Masked ROM versions are indicated with the letter "C" (Z86C03, for example); OTP versions are indicated with the letter "E" (Z86E03, for example).

The on-board dual analog comparators, along with one counter/timer, will be used to implement the ADC routines. The analog comparators are multiplexed with the digital inputs on port pins P31, P32, and P33. This is depicted in Figure 2.

The analog comparators are selected via the P3M register. The comparators share a common reference pin, P33. The input range of the comparators is 0-4V. The input offset voltage is typically 10 mV with V_{cc} at 5.0V. The output of the comparators may be examined by a TM instruction (Test under Mask) on port P3. The outputs also generate an interrupt, based on the falling or rising edge of the comparator output. These outputs also can connect to the P34 and P37 output pins under software control. The PCON register in the extended register file controls this connection (not available on C04/E04 and C08/E08).

The comparators are enabled during HALT mode, but are disabled in STOP mode. Three ADC configurations will be presented:

- PWM Ramp ADC
- Successive Approximation ADC
- RC Ramp ADC

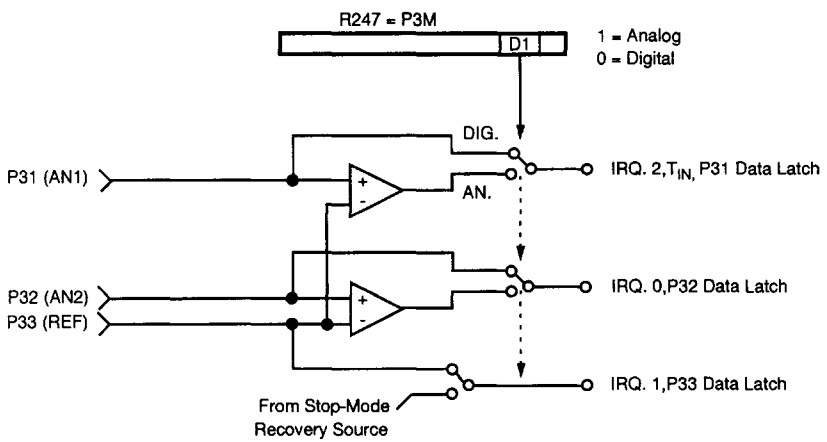


Figure 2. Port 3 Configuration

The software routines were designed around the Z86C04/E04, but can be adapted for other selected Z8 MCUs.

The Zilog CCP emulator (Zilog PN Z86CCP00ZEM) was used to test the routines . These routines are designed to be "generic" in nature, meaning that they can be used with other Z8 MCUs.

PWM RAMP ADC

Figure 3 depicts a Pulse-Width Modulated (PWM) ramp and timer to implement a single-slope ADC. This method has the advantage of being interrupt-driven, so that the processor can perform other tasks while doing a conversion. The conversion time, however, is usually in the tens of milliseconds. Here, a fixed-frequency square wave of variable duty cycle begins at a low value and is steadily increased by incrementing the timer count at the end of each sample period.

The PWM output, (P00), is fed to a RC integrator, which produces a linear ramp at the V_{REF} input of the comparators at P33. Two analog voltages can be measured concurrently using this technique.The analog voltages to be measured are sensed at P31 and P32. The voltage range at these inputs is 0–4V.

Refer to Code Listing 1 at the conclusion of this application note, which shows the single-slope software routine.

Each comparator has its own interrupt level. When the positive-going ramp exceeds the input voltage, the corresponding interrupt will then load the contents of the timer, which is scaled to the measured analog input.

Generate the ramp by incrementing Timer 1's value after each sample period. Incrementing the count from 1 (01H) to 200 (C8H) resets the ramp. The value loaded into the register called DELAY determines the sample period. The conversion speed is determined by the speed of the system clock, the analog-input voltage range, sample frequency, and the resolution of the timer.

In this example, the crystal frequency is 8 MHz, the input voltage range is 0–4V, the sampling frequency is about 4800 kHz, and the resolution is 8-bits. This will yield a maximum conversion time of 40 ms.

If you know that the input voltage range is from 2–4V, then you can restrict the ramp voltage to this range, which yields a faster conversion time. Also, if 8-bits of resolution is overkill, then adding two counts to the timer at the end of each sample period will result in 7 bits of resolution. Adding four counts yields 6 bits.

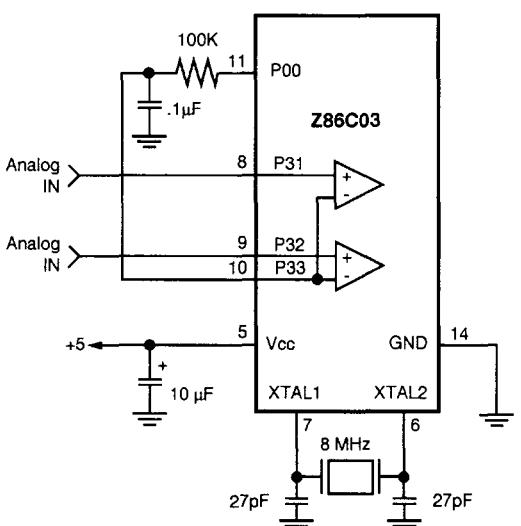


Figure 3. Single Slope ADC Using PWM Ramp

SUCCESSIVE APPROXIMATION ADC

For applications requiring faster conversion times, you want to consider the successive approximation method (see Figure 4). This routine, however, cannot be used in background mode, which means that when a conversion is taking place, the processor must drop what it's doing and call or jump to this routine. However, at the end of a conversion, it can then continue where it left off.

This method uses a Digital-to-Analog Converter (DAC) in its feedback loop. The DAC is comprised of an R2R ladder connected to port P2. When a binary value is output at Port 2, a DC voltage proportional to the binary value appears at pin 1 of the ladder network. The DAC does a binary search on the input voltage. This is achieved by first setting the most significant bit (MSB) of the DAC and testing the comparator output. If the comparator output is zero, then the DAC output for this bit is set to zero. If the comparator output is one, then the DAC output for this bit is set to one. The bits from the output port 2 are individually tested in descending order, performing the same test. When all the bits have been tested, the conversion is complete. The output from the R2R resistor network becomes the comparator's reference voltage. The analog voltage to be measured can be connected to either P31 or P32, the non-inverting inputs of the comparators. The software for this routine is shown in Code Listing 2.

To start the conversion, the MSB of P2 is set, resulting in a voltage of half of V_{CC} at the V_{REF} input of the comparator. If V_{CC} is 5V, then this will be 2.5V. The non-inverting comparator input is then tested. If High, then the analog voltage must be 2.5V–5.0V.

The next bit, P26, is then set, and the input port is tested again. If Low, then this bit is reset. The process continues until all bits of P2 have been tested. The resultant value at P2 is the digital representation of the analog input.

With a crystal frequency of 12 MHz, the conversion time is approximately 70 µs. Even more resolution is available from 10- and 12-bit R2R networks. Of course, more port pins are then needed.

If a track-and-hold function is desired, analog switch CD4016 can be added to the front end. In between samples, the switch is closed, charging a small capacitor.

At the end of each sample period—as determined by timer T1—the switch is opened. Any error induced by trying to convert a fast-changing analog input is eliminated. The value of C is chosen so that the voltage across it tracks the input voltage, does not bleed off during the conversion, and follows the input voltage in between samples. For a sampling rate of 8 kHz, the input frequency must be less than or equal to 4 kHz to prevent distortion due aliasing.

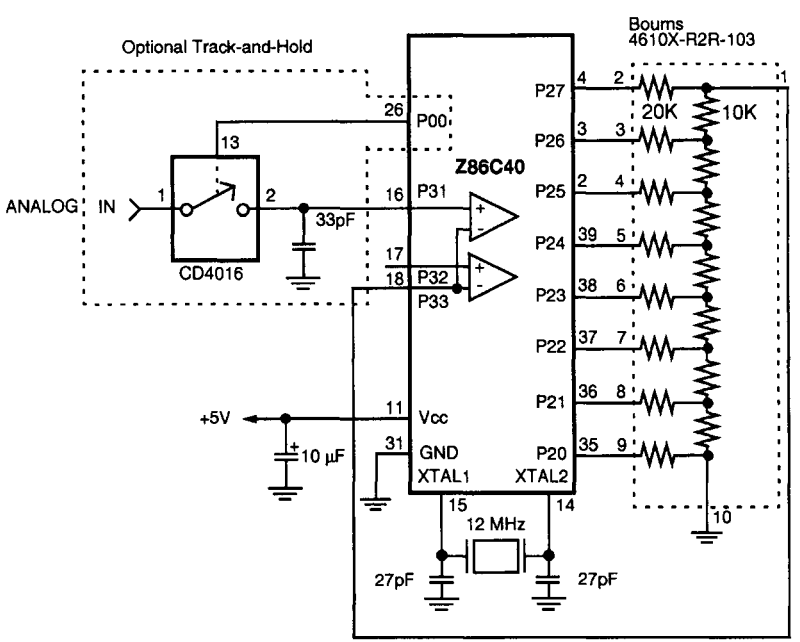


Figure 4. Successive Approximation ADC

RC RAMP ADC

For cost-sensitive applications, consider the low-cost technique that is shown in Figure 5. Since this method uses a logarithmic RC ramp for the comparator reference voltage, the A/D result will not be a linear function of the input analog voltage. For many applications, such as some battery chargers and peak detection circuits, linearity is not essential. If necessary, software compensation by look-up table can be used to linearize the A/D result.

This method uses an externally generated RC ramp and a counter/timer to measure the analog voltage at comparator input P33. The RC ramp is limited to the 4V common mode range of the Z8 comparator. The C1 capacitor is readied for conversion by briefly setting output P00. The C1–P31 node will discharge to 4V, as determined by the R1–R2 voltage divider. When conversion is started, P00 is taken Low so that the C1–P31 node charges toward ground through the parallel combination of R1 and R2.

At the same time the ramp is initiated, counter/timer T1 is started in gate mode. A crystal frequency of 8 MHz will give a counter tick period of 1 ms when the prescaler is set to divide-by-1. When the capacitor voltage at P31 crosses the unknown analog voltage at P33, the comparator will switch, causing the counter/timer T1 to stop.

Code Listing 3 (at conclusion of this application note) shows the routine, which gives a resolution of 8 bits and a maximum conversion time of 256 µs for an oscillator frequency of 8 MHz. The accuracy is ± LSB. If the oscillator frequency is changed, the RC values will need adjustment. The analog input voltage range is 0–4V.

Some applications require accurate detection of a single threshold level. RC value tolerances can be adjusted out with the addition of a single calibration. Simply replace R2 with a potentiometer and series resistor. Calibration is performed by putting the desired threshold voltage at P33 while adjusting the pot for desired result. Use a temperature stable capacitor, such as a ceramic NPO type, if wide temperature variations are expected.

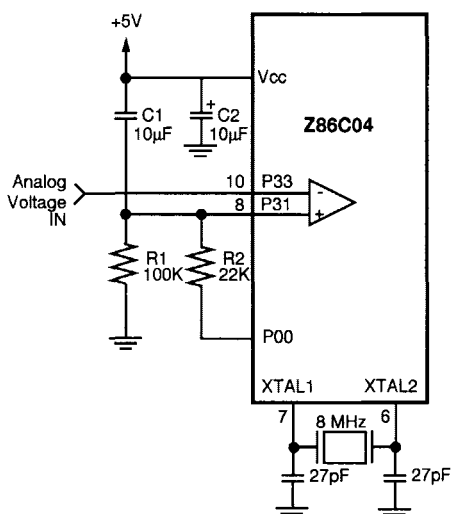


Figure 5. A/D Converter Using RC Ramp

CONCLUSION

You can see from the previous examples that applications requiring analog-to-digital conversions can be achieved without compromising accuracy, speed, or system cost. Design engineers can experiment with the routines for optimum performance. These routines are designed to be functions called from the main program; however, it is possible to modify the program so that it is completely interrupt-driven. This change allows the MCU to handle other tasks in between the timer and comparator interrupts. By adding a software UART routine to the above programs, you can implement a truly low-cost, PC-compatible data acquisition system.

REFERENCE

1. Ciarcia, S., "Analog Interfacing In The Real World," *Ciarcia's Circuit Cellar*, Vol. IV, 1984.

1. Distaso, L.A., "Analog to Digital Conversion Techniques With COPS Family Microcontrollers", National Semiconductor Corp., *National Semiconductor Microcontroller Databook*, 1987.

2. Zilog, *Zilog Discrete Z8 Microcontroller Product Specification Databook*, DC 8318-02, Q3 1994.

3. Zilog, *Zilog Z8 Microcontrollers User's Manual*, UM95Z8000103, 1995.

CODE LISTING 1

```
;*****************************************************************************************
;       SINGLE-SLOPE SOFTWARE ROUTINE
;
;       The PWM-generated ramp ADC uses two timers and counters and an RC integrator to find the
;       unknown input voltage. When the ramp voltage exceeds V_IN, an interrupt is generated. The
;       value in timer T1 is the digital representation of this voltage.
;*****************************************************************************************
delay_hi                        .equ    r4
delay_lo                        .equ    r5
delay                           .equ    rr4
count                           .equ    r6
irq_0                           .equ    1

                                .org    0

                                .word   comp_trip
                                .word   0
                                .word   0
                                .word   0
                                .word   timer_0
                                .word   timer_1

                                .org    0ch

                                di                          ; disable interrupts
                                srp     #0                  ; set reg pointer 0-15h
                                ld      spl,#80h            ; set stack pointer at top of stack
                                ld      p01m,#4             ; internal stack
                                ld      p3m,#3              ; active pull-ups on p2, comp on
                                ld      p2m,#0ffh           ; inputs on p2
                                ld      pre0,#04h           ; divide by one, one-shot
                                clr     t0                  ; t0 sets the sample period
                                ld      pre1,#06h           ; divide by one, one-shot mode
                                ld      count,#4            ; start counter with minimum count
                                ld      t1,count            ; load counter
                                ld      p0,#04h             ; take port pin P02 high to start
                                ld      imr,#31h            ; load interrupt mask register
                                clr     irq                 ; clear interrupt request reg
                                clr     ipr                 ; clear interrupt priority reg
                                ld      tmr,#0fh            ; load and enable both counters
wait_here:                      ei                          ; enable interrupts
                                jr      wait_here           ; wait for interrupts
;-----------------------------------------------------------------------------------------
;       Timer 0 interrupt routine. Here, the count is incremented every 1000 µs, until a maximum
;       count of 240 is reached. The count is then reset, and the integrating capacitor discharges.
;       Port pin P00 is then taken high, and the timers are reloaded and enabled.
;-----------------------------------------------------------------------------------------
timer_0:                        inc     count               ; start with min count
                                cp      count,#240          ; max count?
                                jr      ult,continue        ; if not maximum, reload timer
                                ld      count,#4            ; load count with initial value
delay_loop:                     decw    delay               ; let cap discharge
                                jr      nz,delay_loop       ; wait awhile
continue:                       ld      t1,count            ; load timer with count
                                or      p0,#04h             ; take p02 high
                                or      tmr,#0fh            ; load and enable t1
                                iret                        ; return from interrupt
```

```
;-------------------------------------------------------------------------------------------------
;       Timer 1 interrupt routine. Port pin P00 is toggled low.This sets the duty cycle for the PWM.
;-------------------------------------------------------------------------------------------------
timer_1:                        xor     p0,#04h             ; take p02 low
                                iret                        ; return from interrupt
;-------------------------------------------------------------------------------------------------
;       Comparator interrupt routine. When the ramp voltage at P33 exceeds the input voltage, the
;       program vectors here. The value in register called count is proportional to the input
;       voltage.
;-------------------------------------------------------------------
comp_trip:                      jr      comp_trip           ; stop here
                                .end
```

CODE LISTING 2

```
;*************************************************************************************************
;       SUCCESSIVE APPROXIMATION ADC ROUTINE
;
;      The successive-approximation ADC uses an R2R ladder connected to an output port. The software
;       performs a binary search on the input voltage. At the end of the conversion, P2 contains ;
the digital representation of the input voltage.
;*************************************************************************************************
test_p32        .equ    r4
ring            .equ    r5
results         .equ    r6

                .org    0h

                .word   0
                .word   0
                .word   0
                .word   0
                .word   0
                .word   0

                .org    0ch

init:           di                      ; disable interrupts
                ld      spl,#80h            ; set stack at top of reg file
                ld      p01m,#4             ; set for internal stack
                ld      p3m,#3              ; active on p2, turn on comp
                ld      p2m,#00             ; p2 all outputs
                srp     #0                  ; set pointer to bottom of reg file
                ld      test_p31,#01h       ; test comparator with working reg
                clr     p2                  ; start out at zero
                clr     ring                ; reset ring register
                scf                         ; set carry flag
next_bit:       rrc     ring            ; rotate a one through ring reg
                jr      c,EOC           ; end of conversion if a carry
                or      p2,ring         ; set P2 bit
                nop                     ; let it settle
                tm      p3,test_p32     ; test p32 input
                jr      nz,next_bit     ; if not low, set next bit
                xor     p2,ring         ; reset bit if it went low
                jr      next_bit        ; continue
EOC:            ld      results,p2      ; gets results
                jr      EOC             ; loop here when done
                .end
```

CODE LISTING 3

```
;*****************************************************************************************************
;       RC RAMP ADC
;
;       This routine uses an externally generated ramp and a counter timer to measure the voltage
;       at comparator input P33. This routine gives a resolution of 8 bits and a maximum conversion
;       time of 256 µs for an oscillator frequency of 8 MHz.
;*****************************************************************************************************
result                  .equ    r4
delay                   .equ    r5

irq2                    .equ    4

                        .org    0                   ; start of interrupt vector table

                        .word   0                   ; interrupt 0 vector address
                        .word   0                   ; interrupt 1 vector address
                        .word   0                   ; interrupt 2 vector address
                        .word   0                   ; interrupt 3 vector address
                        .word   0                   ; interrupt 4 vector address
                        .word   0                   ; interrupt 5 vector address

                        .org    0ch                 ; start code execution here

                        di                          ; disable interrupts
                        srp     #0                  ; set reg pointer 0-15h
                        ld      spl,#80h            ; set stack pointer at top of stack
                        ld      p01m,#4             ; internal stack
                        ld      p3m,#3              ; pull-ups on p2, comparators on
                        ld      p2m,#00h            ; outputs on p20-p27
                        ld      pre1,#06h           ; divide by one, one-shot mode
                        clr     t1                  ; clear timer
                        clr     tmr                 ; timer 1 off
                        clr     imr                 ; clear interrupt mask reg
                        clr     irq                 ; clear interrupt request reg
                        clr     ipr                 ; clear interrupt priority reg
start:                  ld      p0,#02h             ; discharge cap
                        clr     delay               ; load delay with discharge delay
delay_loop:             djnz    delay,delay_loop    ; wait until cap discharged
                        clr     p0                  ; take p20 low (start ramp)
                        or      tmr,#0ch            ; load and start timer
poll_irq2:              ei                          ; enable interrupts
                        tm      irq,#irq2           ; test for comparator trip (irq2)
                        jr      z,poll_irq2         ; if not set, continue polling
                        ld      result,t1           ; get count from t1
                        com     result              ; compliment
conversion_done:        jr      conversion_done     ; loop here when done
                        .end?
```

A DC MOTOR CONTROLLER USING THE ZILOG Z86E06 MCU

THIS DC MOTOR CONTROLLER MORE THAN MEETS THE CHALLENGES OF SMALLER PACKAGING AND HIGHER RELIABILITY.

INTRODUCTION

Many applications such as printers, ATM machines, robotics, CD players, and disk drives require DC motor control. Additionally, the demand for smaller packaging and higher reliability creates a need for more integrated motor driver solutions. In the past, a DC motor controller comprised a microcontroller (MCU) and many discrete components. Presently, advanced IC technology reduces the drive electronics to just two chips: an MCU and an integrated motor driver. This application note will present a DC motor controller, which uses the Zilog Z86E06 MCU and the National LMD18200 DC Motor Controller IC, that is both low cost and features a low parts count.

MOTOR CONTROL BASICS

Several DC motor design topologies exist, but certainly the most widely used method is the "H-bridge" configuration. This method uses four Bi-polar Junction Transistor (BJT) or Metal-Oxide Silicon Field Effect Transistor (MOSFET) devices configured in an "H" pattern (see Figure 1). In the center of the "H" is the motor itself. To drive the motor in the forward direction, current flows through Q1 and Q4. To turn the motor in the reverse direction, Q1 and Q4 are turned off, and Q2 and Q3 are energized. External logic is needed to gate the devices. Motor speed is controlled by the average current flowing in the legs. This is regulated by a Pulse-Width Modulation (PWM) drive method, which relies on the duty cycle of a digital output to control the drive voltage to the MOSFETs (see Figure 2). Conventional designs require low-pass filtration to produce a constant DC voltage. Varying the duty cycle of the output varies the DC voltage. This DC voltage would then drive the power MOSFETs. The necessary steering logic is incorporated inside the LMD18200. The PWM is also accepted without the need for an external low-pass filter.

Since the voltage drop across the collector and emitter can reach more than 1V during saturation, high heat dissipation is encountered using BJT devices. MOSFETs, with their intrinsically low Rds (drain to source turn-on resistance), are better suited for motor driver applications. Typically, power MOSFETs need a gate voltage of least 8V to turn on, which is a problem when using an MCU whose outputs swing from 0–5V. Special logic MOSFETs have been developed that have gate turn-on voltages of 5V. This works fine for the lower legs of the "H" motor drive, but what about the upper legs? Unfortunately, the upper legs need a higher gate voltage, due to the fact that the motor's winding resistance raises the MOSFETs source reference above ground potential. A separate DC-DC convertor chip can be used, but this adds more cost and complexity to the design. The LMD18200 solves this problem by having a built-in DC-DC convertor.

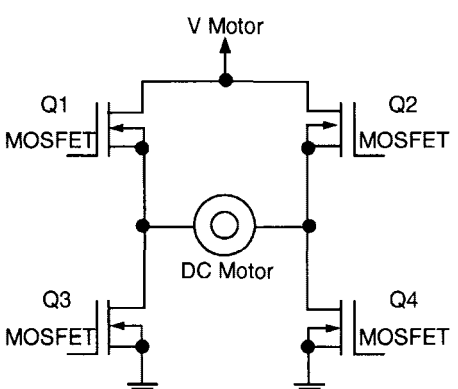


Figure 1. H-Pattern Motor Driver Configuration

HARDWARE DESCRIPTION

The heart of the motor controller is the Zilog Z86E06, an 18-pin, One-Time Programmable (OTP) MCU, which contains 1KB of ROM, 128 bytes of RAM, 14 I/O lines, two timer/counters, and two on-board comparators. A block diagram of the chip is shown in Figure 2. The Z8® MCU is clocked by a 4 MHz ceramic resonator, and the motor is controlled by the National LMD18200. This is a 3A H-Bridge driver IC with direction and braking logic, along with thermal and current sensing of the output drivers. The functional diagram of the LMD18200 is shown in Figure 3.

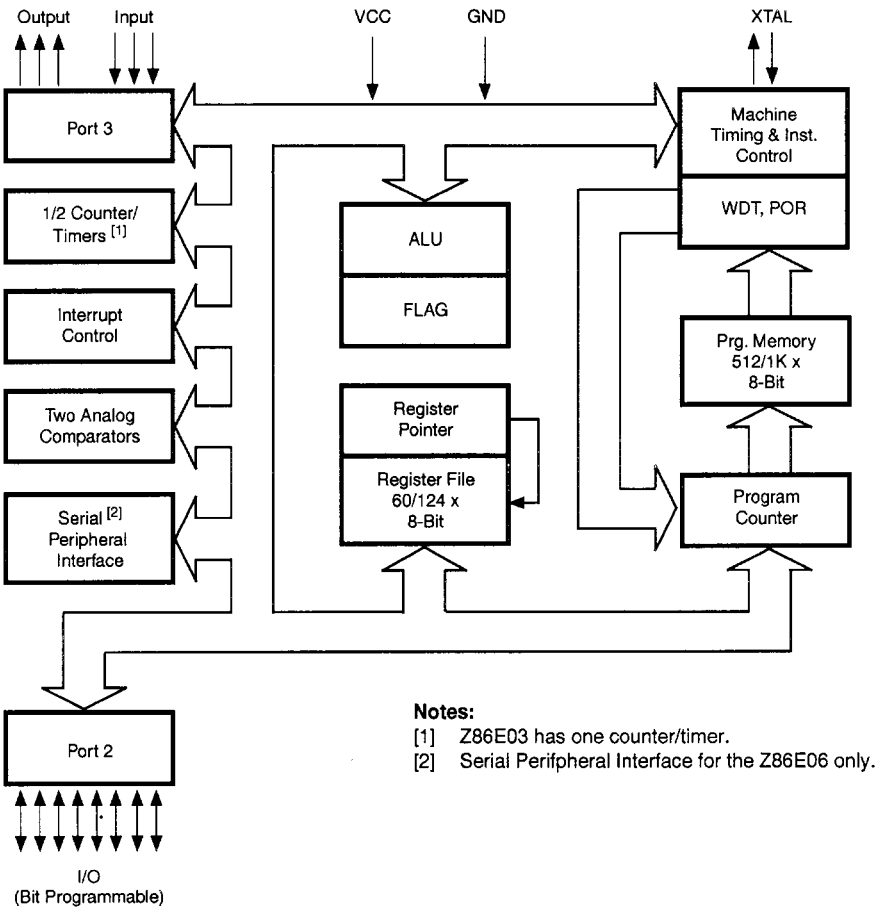


Figure 2. Z86E03/E06 Functional Block Diagram

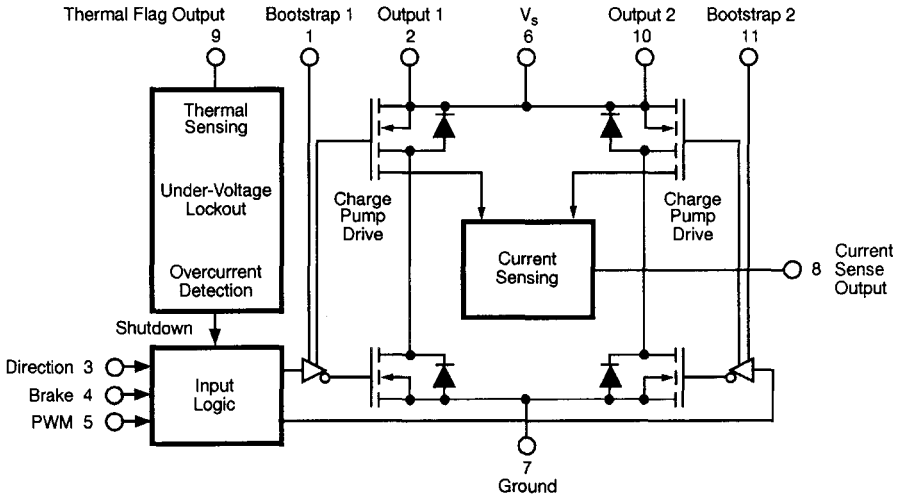


Figure 3. LMD18200 Functional Diagram

Referring to the Z86E03/E06 schematic diagram (see Figure 4), output port pins P34, P35, and P36 of the Z86E06 provide direction, brake, and PWM signals, respectively, to the LMD18200. The current sensing is done with a resistor to ground from pin 8 of the LMD18200. The current output from this pin is typically 377 µA/per A. If the motor is rated at 1A, a resistor value is chosen so that the voltage developed across the resistor does not exceed 4V, which is the maximum input voltage to the comparator on-board the Z86E06. This calculates out to be 10.6K ohms (4V/377µA). For this application, the motor was rated at 12V @ 0.6A. A resistor value of 2.7K ohms was chosen to give a maximum voltage across the resistor of approximately 2.5V at all speeds. This voltage is sensed by one of the Z8® MCU's on-board comparators. The reference pin for the comparators (P33) is biased at 3.3V through a voltage divider.

Any significant load on the motor will increase the motor current, thereby increasing the voltage across the current sense resistor. If the voltage across the sense resistor exceeds 3.3V, a comparator interrupt will be generated, and the BRAKE line to the LMD18200 will be activated, stopping the motor. When the load condition of the motor is removed, pressing the STOP push button reactivates the motor.

The temperature flag pin (9) of the LMD18200 is connected to P32 of the Z8 MCU. This is an open-collector output, therefore it is pulled up to +5V with a 10K ohm resistor. The function of this pin is to interrupt the processor when the junction temperature of the LMD18200 exceeds 145 degrees C. This is an active-low output, which will trip the Z8 MCU comparator at P32. A push button connected to P20-P22 of the Z8 MCU provides the speed and brake control.

The SPDT toggle switch connected to P23 provides the direction (high is forward, low is reverse). Direction changing can only be done in a STOP condition. Status LEDs show the direction and stop conditions. The STOP LED (red) is connected to P24. The FORWARD (yellow), and REVERSE (green) LEDs are connected to P25 and P26, respectively.

The motor terminals are connected between the OUT1 and OUT2 terminals of the LMD18200. The 0.1 µF ceramic capacitors are connected between the bootstrap pins (1,11) and motor output pins (2,10). The Vs pin is connected to +12V. This line is bypassed with a 220 µF and 0.1 µF capacitors.

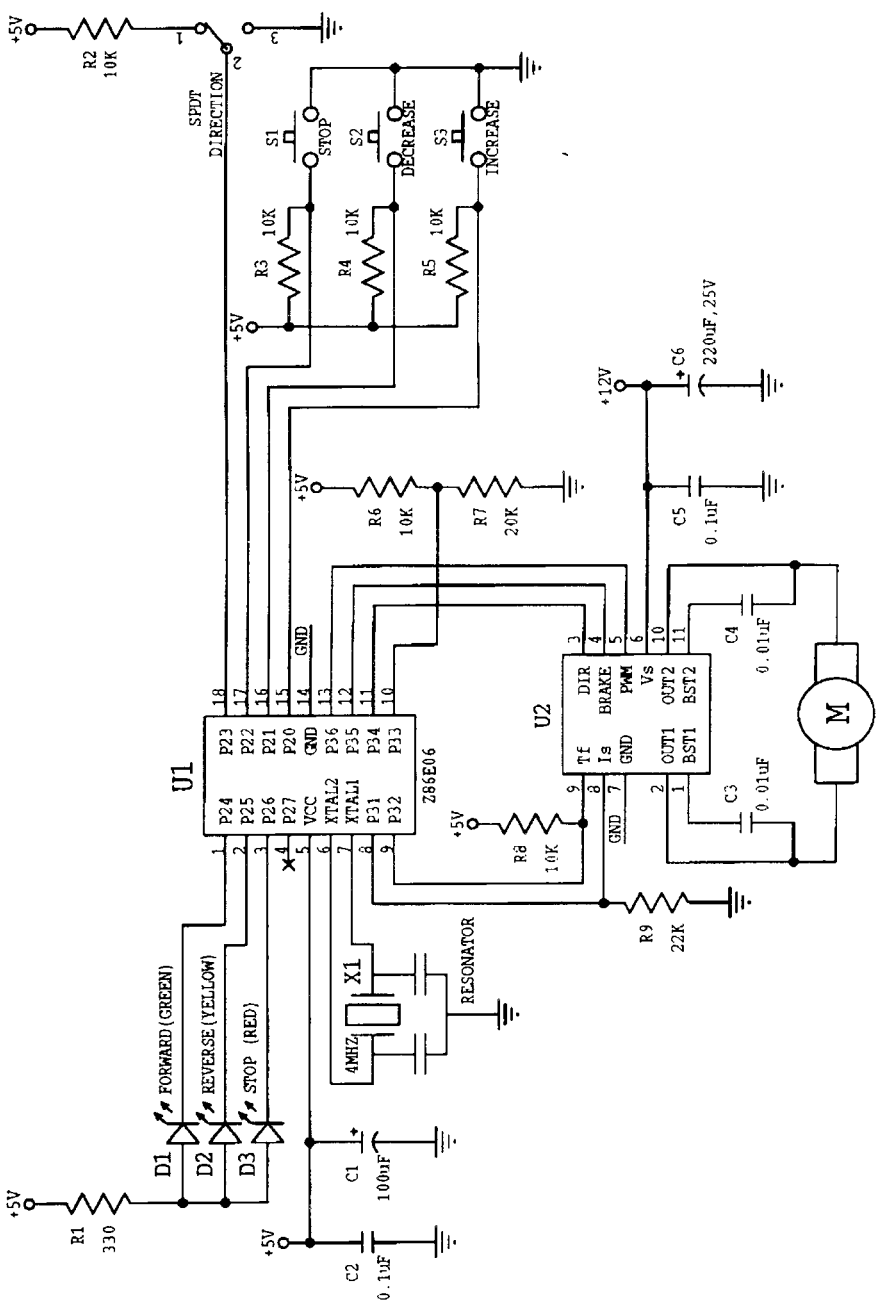


Figure 4. DC Motor Controller Schematic

SOFTWARE DESCRIPTION

The motor controller software flowchart is shown in Figure 5. After initialization, the controller waits for a timer T0 interrupt. The timer interval is set for about 500 µs. This was chosen to provide a 2000 Hz. sample rate for the PWM.

The SAMPLE interrupt service routine essentially handles two functions:

- The PWM Output
- Sensing and Debouncing of the Push Button

The comparator interrupts sense overload and over-temperature conditions. (Refer to Code Listing at the conclusion of this application note.)

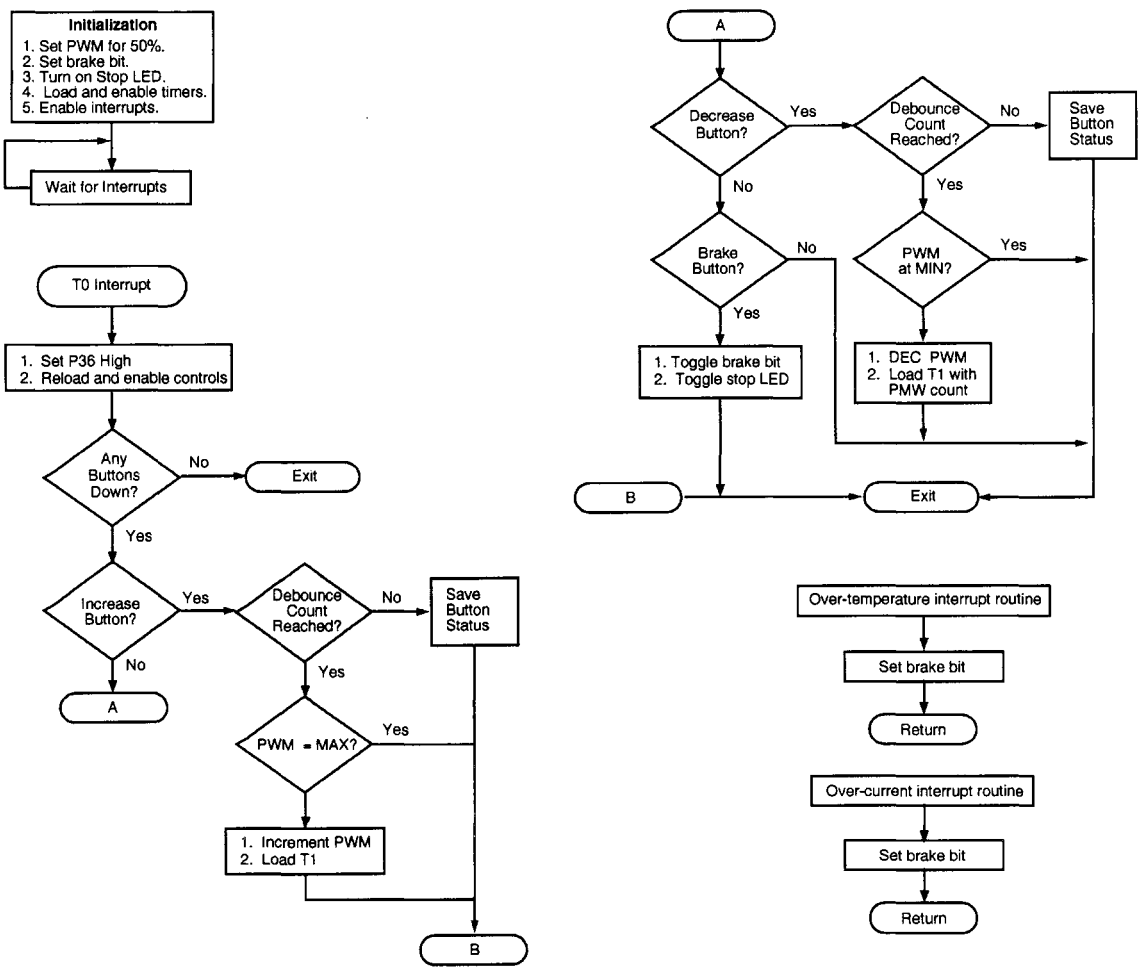


Figure 5. DC Motor Flowchart

Refer to Figure 6 for PWM Waveforms. The initial PWM is set for a 50-percent duty cycle. The initial state is STOP, as indicated by the red LED. The motor may be taken out of STOP by momentarily pressing the STOP button. The motor will then turn at a speed determined by the PWM value and either clockwise or counterclockwise, as determined by the DIRECTION switch. The duty cycle of the PWM is controlled by timer T1. This is loaded with a value contained in register PWM. The timer is configured to count down, then stop when it reaches zero. When it hits terminal count, it toggles port pin P36 from high to low. The timer is loaded with its initial value, and P36 is taken high on the next pass of the T0 interrupt interval. The timer T0 sets the sampling rate, in this case 2000 Hz.

Pressing the INCREASE push button increases the duty cycle of the PWM, while the DECREASE push button decreases the duty cycle of the PWM. The maximum limits are 240/256 (93 percent) and 32/256 (12.5 percent), respectively. Pressing the STOP button toggles the BRAKE output to the LMD18200, and either brakes or enables the motor, depending on the present state.

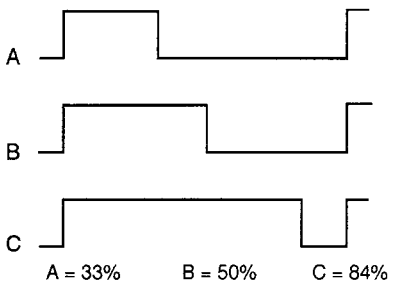


Figure 6. PWM Waveforms Duty Cycle

The temperature flag input to the Z8® MCU senses the condition of this signal from the LMD18200. This pin goes active low when the junction temperature of the motor controller reaches 145 degrees C. The assertion of this pin (active low) causes the compartor to trip at P32, interrupting the processor. The OVER_TEMP interrupt service routine sets the BRAKE output to high, disabling the motor. This condition will exist until the THERMAL FLAG bit goes high again.

REMOTE CONTROL

Instead of push button switches to control the functions and speed of the motor, an I^2C or asynchronous communications protocol can be added for remote control.

REFERENCES

1. Chuck McManis, "A PIC-based Motor Speed Controller," *Circuit Cellar Ink*, July, 1995.
2. Jeff Bachiacchi, "Creating The Smart-MD," *Circuit Cellar Ink*, September–October,1995.
3. National Semiconductor, LMD18200 datasheet.
4. Zilog, *Zilog Discrete Z8 Microcontrollers Product Specifications Databook*, DC 8318-02.
5. Zilog, Zilog Z8 Microcontroller User's Manual, UM95Z800103.

CODE LISTING

```
asmS8 version 2.1
Tue Feb 27 02:59:25 1996        b:\testdnm
LOC      OBJ                LINE# --- SOURCE ---
                                1 ;--------------------------------------------------------------------------------
                                2 ;      This program is intended to control a DC motor with a Zilog
                                3 ;      microcontroller. The Z8 produces a PWM (Pulse-Width Modulation) signal
                                4 ;      that drives a National LMD18200 H-bridge motor driver. This allows
                                5 ;      full directional and speed control of the motor. The speed,
                                6 ;      direction, and stop functions are controlled by pushbutton switches.
                                7 ;      A status LED indicates Forward (Green), Reverse (Yellow), and Stop
                                8 ;      (Red) conditions. A voltage feedback from the H-bridge allows the
                                9 ;      microcontroller to monitor the current through the motor. This is
                               10 ;      implemented with a series resistor from the Is pin of the H-bridge,
                               11 ;      and the voltage developed across the resistor is proportional to the
                               12 ;      current through the motor. This voltage is fed back to one of the Z8's
                               13 ;      on-chip analog comparators.
                               14 ;
                               15 ;
                               16 ;
                               17 ;      Z86E06 pin assignments
                               18 ;      ----------------------
                               19 ;                  1              18
                               20 ;                   ---------------
                               21 ;FORWARD LED  - | P24        P23 | -  DIR SWITCH
                               22 ;               |                |
                               23 ;REVERSE LED  - | P25        P22 | -  STOP PUSHBUTTON
                               24 ;               |                |
                               25 ;   STOP LED  - | P26        P21 | -  SPEED DECREASE PUSHBUTTON
                               26 ;               |                |
                               27 ;        N/C  - | P27        P20 | -  SPEED INCREASE PUSHBUTTON
                               28 ;               |                |
                               29 ;      +5V    - | Vcc        GND | -  GND
                               30 ;               |                |
                               31 ;      OSC    - | Xtal1      P36 | -  PWM OUT
                               32 ;               |                |
                               33 ;      OSC    - | Xtal2      P35 | -  STOP
                               34 ;               |                |
                               35 ;      Vref   - | P33        P34 | -  DIRECTION
                               36 ;               |                |
                               37 ;      Tf     - | P32        P31 | -  Is
                               38 ;                ---------------
                               39 ;                9              10
                               40 ;
                               41 ;--------------------------------------------------------------------------------
                               42
      abs 00000004             43 bounce           .equ   r4
      abs 00000005             44 count            .equ   r5
      abs 00000006             45 key_cnt          .equ   r6
      abs 00000007             46 key_temp         .equ   r7
      abs 00000008             47 temp_l           .equ   r8
      abs 00000009             48 pwm              .equ   r9
      abs 0000000a             49 make             .equ   r10
      abs 0000000b             50 STATE            .EQU   R11
      abs 0000000c             51 temp_led         .equ   r12
      abs 0000000e             52 delay_hi         .equ   r14
      abs 0000000f             53 delay_lo         .equ   r15
      abs 0000000e             54 delay            .equ   rr14
                               55
  0000000000000000001          56 increase         .equ   01h
  0000000000000000002          57 decrease         .equ   02h
  0000000000000000003          58 brake            .equ   03h
  0000000000000000008          59 dir_sw           .equ   08h
  0000000000000000007          60 switches         .equ   07h
  0000000000000000020          61 min              .equ   20h
  00000000000000000e0          62 max              .equ   0e0h
  0000000000000000001          63 irq0             .equ   01h
  0000000000000000004          64 irq2             .equ   04h
  0000000000000000010          65 irq4             .equ   10h
  0000000000000000020          66 brakes_on        .equ   20h
  0000000000000000040          67 stop_led         .equ   40h
```

```
0000000000000000020    68 reverse_led     .equ   20h
0000000000000000010    69 forward_led     .equ   10h
0000000000000000000    70 stopped         .equ   00h
0000000000000000001    71 start_up        .equ   01h
0000000000000000002    72 running         .equ   02h
                       73 ;---------------------------------------------------------------------------
                       74 ;               INITIALIZATION
                       75 ;---------------------------------------------------------------------------
00000000               76                 .org   0000h
                       77
00000000 Wwww          78                 .word  over_temp
00000002 Wwww          79                 .word  no_irq
00000004 Wwww          80                 .word  over_current
00000006 Wwww          81                 .word  no_irq
00000008 Wwww          82                 .word  sample
0000000a Wwww          83                 .word  no_irq
                       84
0000000c               85                 .org   000ch
                       86
0000000c 8f            87                 di                          ; disable int
0000000d 3100          88                 srp    #0                   ; lowest bank
0000000f e6f60f        89                 ld     p2m,#0fh             ; inputs on p20-p23, outputs on p24-p27
00000012 e602bf        90                 ld     p2,^C #stop_led      ; load with initial values
00000015 e6f702        91                 ld     p3m,#2               ; open drain on P2, comparators on
00000018 e6f804        92                 ld     p01m,#04h            ; int stack
0000001b e6ff80        93                 ld     sp1,#80h             ; stack at highest ram location
0000001e b0fe          94                 clr    sph                  ; clear stack pointer high byte
00000020 b0fa          95                 clr    irq                  ; clear  int request reg
00000022 e6f504        96                 ld     pre0,#04h            ; load prescaler 0 with /1, one-shot
00000025 b0f4          97                 clr    t0                   ; set timer TC for period of 0.5 mS
00000027 e6fb10        98                 ld     imr,#irq4            ; set interrupt levels
0000002a e6f306        99                 ld     pre1,#06h            ; one shot
0000002d e60360       100                 ld     p3,#60h              ; brakes on, pwm high
00000030 bc00         101                 ld     STATE,#stopped       ;
00000032 b0f9         102                 clr    ipr                  ;
00000034 b0ee         103                 clr    delay_hi             ;
00000036 b0ef         104                 clr    delay_lo             ;
00000038 9c80         105                 ld     pwm,#80h             ; start with pwm = 50%
0000003a 99f2         106                 ld     t1,pwm               ; load timer with pwm value
0000003c e6f18f       107                 ld     tmr,#8fh             ; load and enable t0,t1
0000003f e6fa80       108                 ld     irq,#80h             ; rising on irq2, falling on irq0
00000042 9f           109 delay_loop:     ei                          ; enable interrupts
00000043 8bfd         110                 jr     delay_loop           ; wait for interrupts
                      111 ;---------------------------------------------------------------------------
                      112 ;               OVER-TEMPERATURE INTERRUPT ROUTINE
                      113 ;---------------------------------------------------------------------------
00000045 460320       114 over_temp:      or     p3,#brakes_on        ; set BRAKE line high
00000048 bc00         115                 ld     STATE,#stopped       ;
0000004a e602bf       116                 ld     p2,^C #stop_led      ;
0000004d e6fb10       117                 ld     imr,#irq4            ;
00000050 bf           118                 iret                        ; return from interrupt
                      119 ;---------------------------------------------------------------------------
                      120 ;               OVER-CURRENT INTERRUPT ROUTINE
                      121 ;---------------------------------------------------------------------------
00000051 460320       122 over_current:   or     p3,#brakes_on        ; set BRAKE line high
00000054 bc00         123                 ld     STATE,#stopped       ;
00000056 e602bf       124                 ld     p2,^C #stop_led      ;
00000059 e6fb10       125                 ld     imr,#irq4            ;
0000005c bf           126                 iret                        ;
                      127 ;---------------------------------------------------------------------------
                      128 ;               TIMER 0 INTERRUPT ROUTINE
                      129 ;
                      130 ;               This routine performs the following functions:
                      131 ;
                      132 ;               1) Sets sample rate for PWM at 2000 Hz
                      133 ;               2) Tests for key closures and checks direction switch
                      134 ;---------------------------------------------------------------------------
0000005d a6eb00       135 sample:         cp     STATE,#stopped       ; check if stopped
00000060 eb**         136                 jr     ne,test_start_up;
00000062 760208       137                 tm     p2,#dir_sw           ; test direction switch
00000065 6b**         138                 jr     z,ccw                ; if low, then reverse
00000067 460310       139                 or     p3,#10h              ; if high, then forward
```

```
0000006a ccef            140                 ld    temp_led,^C #forward_led
0000006c 8b**            141                 jr    continue                ;
0000006e 5603ef          142 ccw:            and   p3,^C #10h              ; take direction bit low
00000071 ccdf            143                 ld    temp_led,^C #reverse_led
00000073 a6eb01          144 test_start_up:  cp    STATE,#start_up         ;
00000076 eb**            145                 jr    ne,continue             ;
00000078 80ee            146                 decw  delay                   ;
0000007a eb**            147                 jr    nz,continue             ;
0000007c e6fb15          148                 ld    imr,#irq4+irq2+irq0;
0000007f bc02            149                 ld    STATE,#running          ;
00000081 460340          150 continue:       or    p3,#40h                 ; take pwm high
00000084 e6f18f          151                 ld    tmr,#8fh                ; load and enable timer
00000087 660207          152 key_scan:       tcm   p2,#switches            ; any switches pressed?
0000008a 6b**            153                 jr    z,exit                  ; no, then exit
0000008c 8802            154                 ld    temp_1,p2               ; get switch data
0000008e 5c03            155                 ld    count,#3                ; load counter
00000090 6e              156 key_loop:       inc   key_cnt                 ; inc key count
00000091 df              157                 scf                           ; set carry flag
00000092 c0e8            158                 rrc   temp_1                  ; rotate right
00000094 7b**            159                 jr    c,no_keys               ; any carry?
00000096 a276            160                 cp    key_temp,key_cnt        ; same key?
00000098 eb**            161                 jr    ne,load_keys            ; not the same
0000009a 4e              162                 inc   bounce                  ; increment bounce counter
0000009b a6e480          163                 cp    bounce,#80h             ; bounce = 127 ?
0000009e 7b**            164                 jr    ult,load_keys           ; no action if less
000000a0 a6e601          165                 cp    key_cnt,#increase       ; increase key?
000000a3 eb**            166                 jr    ne,try_decrease         ; no, try decrease key
000000a5 a6e9e0          167                 cp    pwm,#max                ; are we at maximum pwm?
000000a8 6b**            168                 jr    eq,exit                 ; yes, don't increment
000000aa 9e              169                 inc   pwm                     ; increment pwm (increase speed)
000000ab 99f2            170                 ld    t1,pwm                  ; load timer with pwm value
000000ad 8b**            171                 jr    exit                    ; exit routine
000000af a6e602          172 try_decrease:   cp    key_cnt,#decrease       ; decrease key?
000000b2 eb**            173                 jr    ne,try_brake            ; try brake key
000000b4 a6e920          174                 cp    pwm,#min                ; is pwm at minimum?
000000b7 3b**            175                 jr    ule,exit                ; if yes, don't decrement
000000b9 00e9            176                 dec   pwm                     ; decrement pwm (decrease speed)
000000bb 99f2            177                 ld    t1,pwm                  ; load new pwm value
000000bd 8b**            178                 jr    exit                    ; exit
000000bf a6e603          179 try_brake:      cp    key_cnt,#brake          ; brake button down?
000000c2 eb**            180                 jr    ne,exit                 ; exit if not
000000c4 a6eaff          181                 cp    make,#0ffh              ; button still down?
000000c7 6b**            182                 jr    eq,exit_1               ; yes - take no action
000000c9 b60320          183                 xor   p3,#20h                 ; toggle brake bit
000000cc acff            184                 ld    make,#0ffh              ; set make flag
000000ce 760320          185                 tm    p3,#20h                 ; brake high?
000000d1 eb**            186                 jr    nz,test_dir             ; test dir sw if brakes are on only
000000d3 c902            187                 ld    p2,temp_led             ; load direction led
000000d5 bc01            188                 ld    STATE,#start_up         ; motor turning
000000d7 8b**            189                 jr    exit_1                  ; exit
000000d9 bc00            190 test_dir:       ld    STATE,#stopped          ;
000000db e6fb10          191                 ld    imr,#irq4               ;
000000de c802            192                 ld    temp_led,p2             ; get led data
000000e0 e602bf          193                 ld    p2,^C #stop_led         ; turn on stop led
000000e3 8b**            194                 jr    exit_1                  ;
000000e5 78e6            195 load_keys:      ld    key_temp,key_cnt        ; transfer key data
000000e7 b0e6            196                 clr   key_cnt                 ; clear key register
000000e9 bf              197                 iret                          ; return to interrupt
000000ea 5aa4            198 no_keys:        djnz  count,key_loop          ; tested all keys?
000000ec b0ea            199 exit:           clr   make                    ;
000000ee b0e4            200 exit_1:         clr   bounce                  ; clear key registers
000000f0 b0e6            201                 clr   key_cnt                 ;
000000f2 b0e7            202                 clr   key_temp                ;
000000f4 b0e5            203                 clr   count                   ;
000000f6 bf              204 no_irq:         iret                          ; return from interrupt
                         205
                         206                 .end
```

ZILOG Z8 I/O EXPANDER MODULE

ROMLESS MODE CAN BE USED WITHOUT SACRIFICING THE NORMAL I/O FUNCTION OF PORT0 AND PORT1, AND WITHOUT ADDING ADDITIONAL HARDWARE COMPLICATIONS TO THE TARGET APPLICATION.

INTRODUCTION

The Zilog Z8® MCU family of devices supports a ROMless mode that enables the Z8 code to be executed from an external EPROM. Supporting ROMless mode is relatively easy, requiring only minimal additional hardware. Supporting ROMless mode *and* I/O functions at Port0 and Port1, on the other hand, is a bit more complicated. The ROMless mode configures Port0 and Port1 as address and data lines that are used to interface with the EPROM, making the ports unavailable as general input and output signals. Logic must be added to decode an external memory address; however, adding more logic results in additional hardware complications for the target application.

This application note explains how ROMless mode may be used *without sacrificing* the normal I/O function of Port0 and Port1, and *without adding* additional hardware complications. The Z8 I/O Expander Module described here provides the necessary additional logic used to address an output latch and an input buffer—without further complicating the hardware on the target board with extra circuitry.

Description

The Z8 I/O Expander Module is a small surface mount board developed to make ROMless mode easy to implement (see Figure 1). The decoding logic is integrated with the output latch and an input buffer with a 27C256 EPROM socket. All of the logic is self contained so that the target board can be designed as though Port0 and Port1 are used as input and output lines.

The I/O Expander connects to the target board via a miniature 40-pin connector.

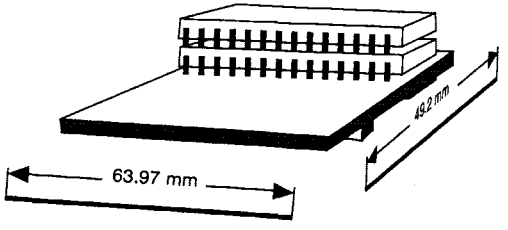


Figure 1. I/O Expander Module (approximate size)

Purpose

The I/O Expander provides a desirable alternative to using an emulator because of size constraints associated with the emulator, ICE pod, and extra power supply. The I/O Expander may be particularly helpful if the prototype is being evaluated by the end customer or a testing agency. In addition, the I/O Expander will improve reliability while enabling the product to be self-contained in the product enclosure.

Retro Workshop 果粉工作室

THEORY OF OPERATION

Refer to the I/O Expander schematic (Figures 3 and 4) for IC references. The I/O Expander board provides a socket for the 27C256 EPROM and its support 74HC374 latch for decoding the upper address lines. A 74HC139 is used to decode address 8000H for both the input and output strobe. Addresses 0 - 7FFFH are decoded by U4B-12 and provide the chip select for the EPROM. Address 8000H is used for both the output latch 74HC374 and input buffer 74HC541, where the R/W signal selects whether U4A-4 or U4A-5 is asserted Low, which depends on whether a Write or a Read function is being performed. The I/O Expander requires all of Port0 to be inputs and all of Port1 to be outputs.

The I/O Expander plugs into the target board using a 40-pin (.050 pin spacing) surface mount connector. The signal pinout was arranged so that when the I/O Expander is removed from the target board a special straight-through shorting block may be plugged in to connect all the pins straight across 1–2, 2–4, and so on. Pins 33 through 40 are not connected by the shorting block since those lines provide the signals from the Z8® MCU to the decoder logic on the I/O expander.

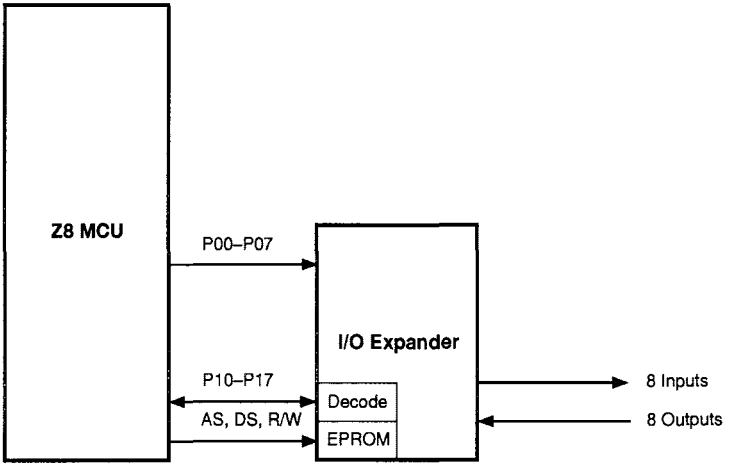


Figure 2. I/O Expander Module Block Diagram

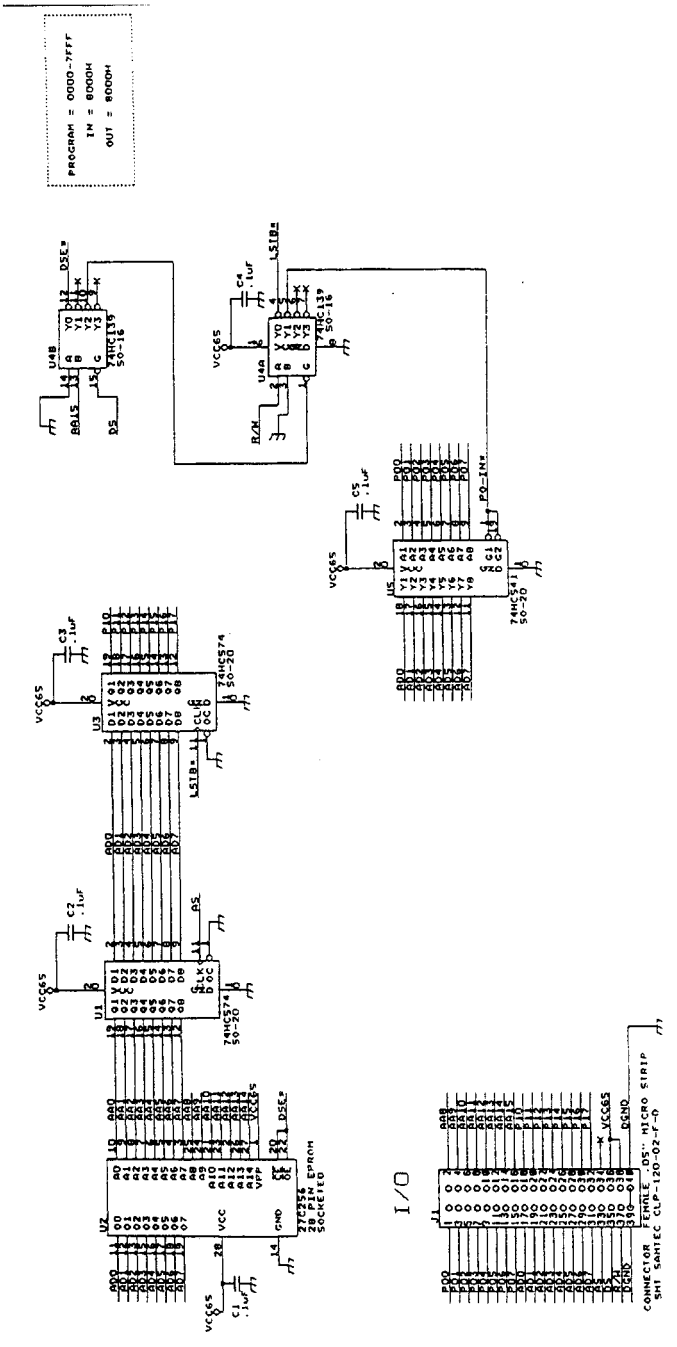


Figure 3. I/O Expander Module Schematic

The schematic (shown in Figures 4 and 5) of the Z89175 FLASH Digital Telephone Answering Device (DTAD) with Caller Identification depicts the I/O Expander implemented in a typical application.

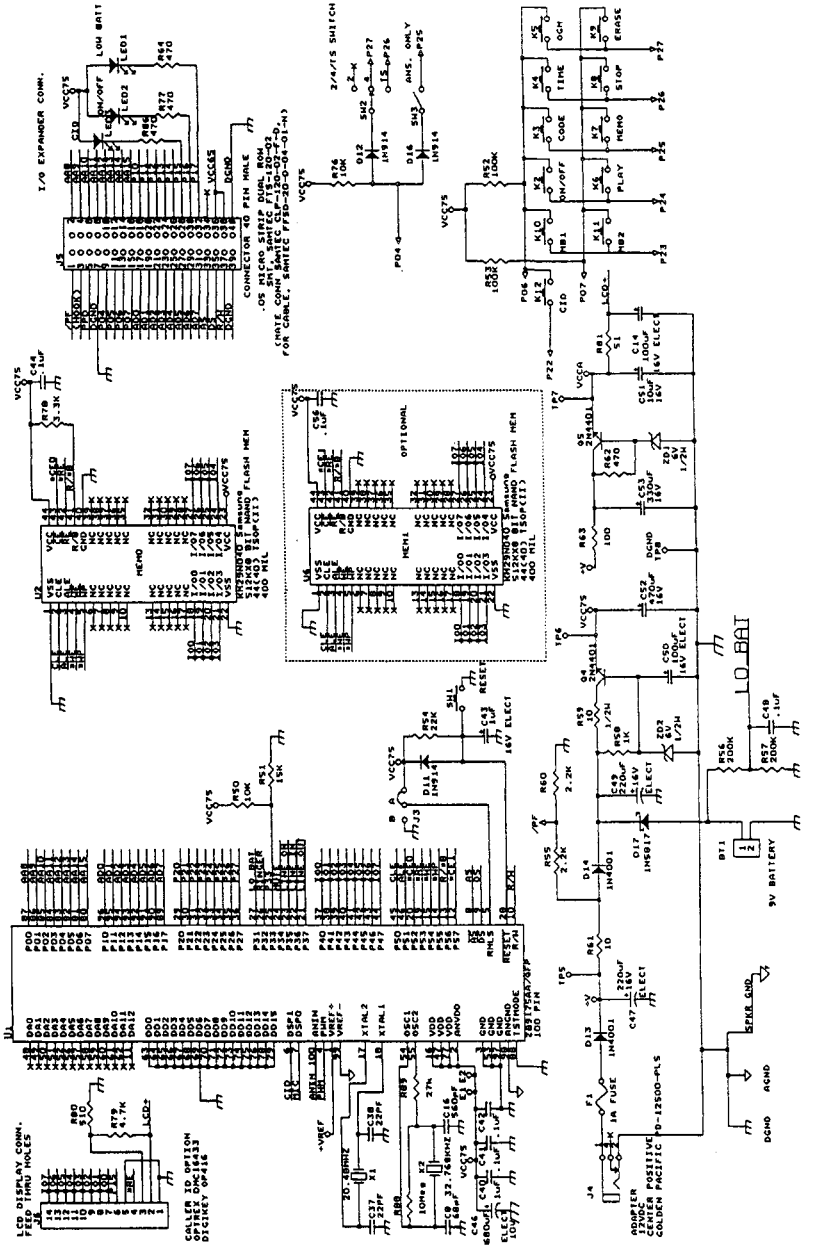


Figure 4. Z89175 FLASH DTAD with Caller ID and I/O Expander Module, Part 1

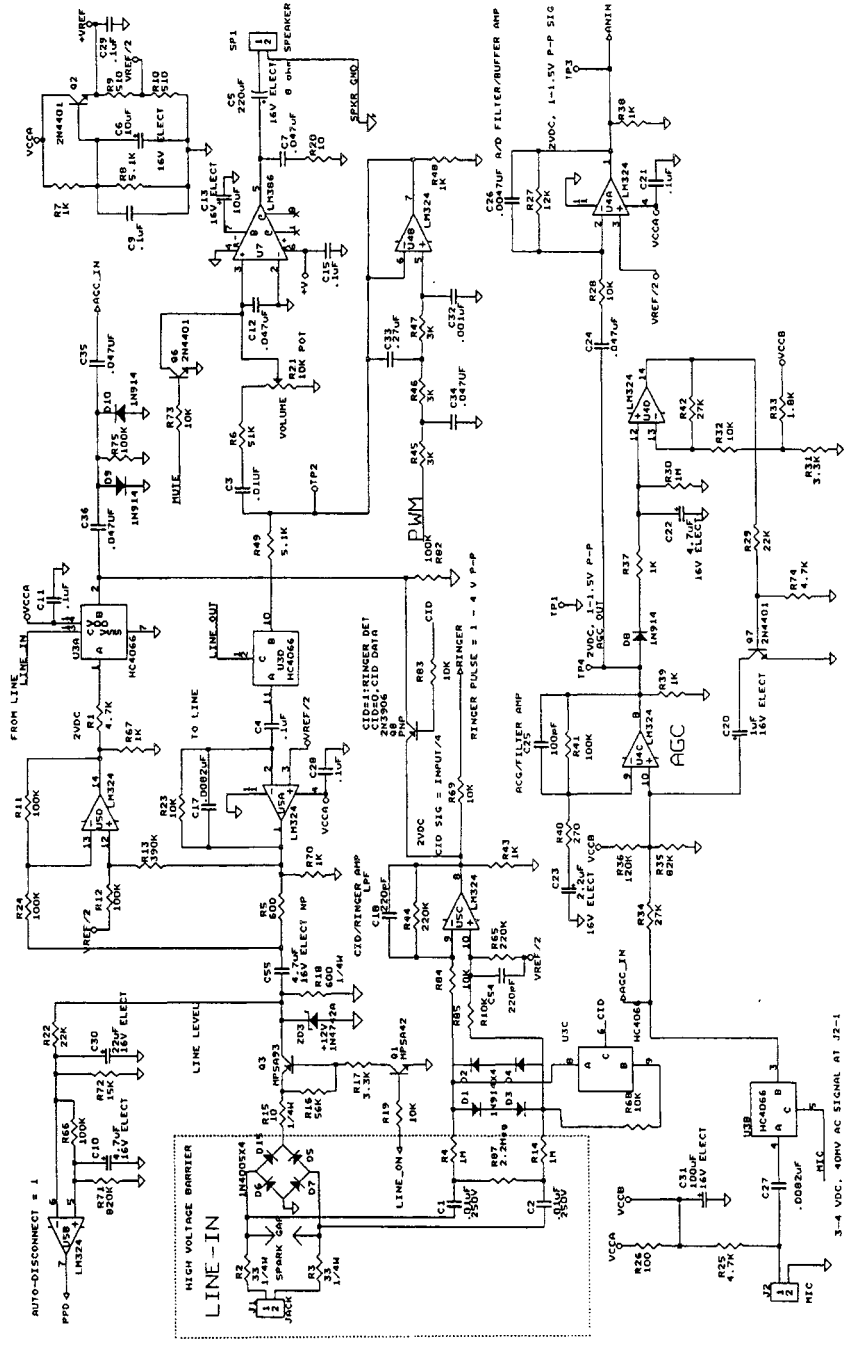


Figure 5. Z89175 FLASH DTAD with Caller ID and I/O Expander Module, Part 2

HARDWARE ISSUES

EPROM access times in the 150–200 ns range are suitable for crystal clock speeds of 10 MHz for most applications. (Reference the appropriate Zilog data sheets for the exact access time requirement for the particular device being used.)

You should carefully consider compatibility of the I/O Expander ports compared to the Z8® MCU ports. The 74HC541 IC used for the inputs does not have an autolatch feature like the Z8 MCU; therefore, if an input is not used and left open it may cause oscillations that could result in noise in adjacent circuits. Any unused inputs should be connected to V_{CC} or GND.

Also, consider that the 74HC374 used for the outputs has a higher output drive capability than the Z8. The hardware should be designed to be within the specifications of the output drive capabilities for the Z8 MCU.

The I/O Expander is hard-wired so that Port0 is all inputs and Port1 is all outputs. The outputs are driven by a 74HC374, which has totem pole outputs. The totem pole outputs should not present a problem in most cases, if the target design uses open-drain on Port1.

Note: If the port was used to scan a keyboard matrix, do not press more than one key at a time unless isolation diodes are used.

Power Consumption Effects

The additional power consumption from the EPROM and I/O Expander should not affect most designs. Most of the extra current (approximately 10 mA) will be used by the EPROM. The three CMOS parts are very low power. The quiescent current on the 74HC374, for example, is about 160 µa (worst case).

The most notable impact the I/O Expander will have on power consumption will be when the application is in a Low-Power mode. Increased power consumption caused by the I/O Expander can be minimized by setting the outputs High or Low, so that minimal current is drawn during either Halt or Stop mode. If the application uses Stop mode, the current from the EPROM will be equal to the standby current listed in the particular Zilog data sheet for the EPROM being used. (A typical value might be 100–250 µa.) For Halt Mode, the current should be significantly less than the active current of the EPROM, since the EPROM will be in standby most of the time. In applications where the Halt Mode is automatically exited by a programmed interrupt, then the standby time will depend on the interrupt rate and the processing time after each interrupt.

SOFTWARE ISSUES

The I/O Expander takes some additional code space, but this is reduced somewhat when the `EXTMEMORY` equate is set to 0 for a mask release. (Refer to the "Program Listings" section, which follows.) In this case, the `InputPort1` and `OutputPort0` subroutines reduce to a single output instruction, the equivalent of what it would be without the I/O Expander. Reading or writing to these ports will require two instructions instead of one.

If the Z8® MCU is a part that includes a DSP processor, the assigned engineer should verify that the burned-in DSP code is the correct version for the intended application. If there was a recent DSP update, that version may not be available in a ROMless part.

The customer should test with both the I/O Expander and the emulator to ensure that there is nothing that could result in a problem with the masked part. This is especially important if the intended masked DSP code is not the version in the ROMless part used with the I/O Expander. In this case, the final software verification should be performed with the emulator.

OutPutPort1 Timing Considerations

When the foreground software uses the `OutputPort1` routine to change the port setting, the exact timing could be affected by the same `OutputPort1` routine used in the interrupt handler.

When the `OutputPort1` routine is called, the changes take effect immediately. If only the "port1save" register is modified, the change will take effect on the next system interrupt. This simpler approach can be used where timing is not critical.

For applications requiring critical timing sequences, the user can temporarily disable interrupts.

SUMMARY

The I/O Expander is useful in applications where Port0 and Port1 are needed by the application as I/O and ROMless mode is also desired. The small size of the module, combined with the software support provided in this application note, makes implementation easy.

The I/O Expander is implemented on a Z89175 Evaluation Kit Board (Z8917500CZ0) that demonstrates digital answering machine functions with FLASH memory and Caller ID. For more information, contact a Zilog sales office.

PROGRAM LISTINGS

The I/O Expander software support includes Port0 and Port1 initialization, input and output subroutines, and some example code that demonstrates how the I/O Expander is used. The names of registers are the same as those used in the Z89175 Evaluation Kit (Z8917500ZC0) application code.

A conditional assembly is used to assemble the code to initialize Port0 and Port1 as address and data if the EXTMEMORY equate is = 1. If the equate is = 0, then the code is assembled to initialize Port0 as an input and Port1 as an output.

```
EXTMEMORY.equ    1                          ; 1=I/O Expander. 0=silicon or emulator

      srp #%F0                              ; Point to control registers
      .if     EXTMEMORY
      ld      r8,#%96                       ; P01M, p0/p1=add/data lines
      .else
      ld      r8,#01000101B                 ; P01M, p0/p1=outputs
      .endif
```

OutputPort1 Subroutine

The OutputPort1 subroutine is used to output a new value to the port. A copy of the output port is always kept in the "port1save" register. Any changes to the port require either an AND or an OR instruction to be performed on the register, then the OutputPort1 subroutine can be called to update the port. If the code is running with the I/O Expander, then the output is performed by an external address write. To run with the emulator or for the mask release, the output is performed by writing to the port directly with a single instruction. The InportPort0 subroutine works in a similar fashion.

```
extadd  .equ   %8000                        ; External Address

OutputPort1:
      .if     EXTMEMORY                     ; Output to Port1
      ld      r0,#^HB(extadd)               ; Point to External I/O
      ld      r1,#^LB(extadd)
      ld      r7,port1save                  : Get New Data
      lde     @rr0,r7                       ; Output Data to "Fake" Port1
      ret

      .else
      ld      port1,port1save               ; Output Real Port1 Data
      ret
      .endif
```

InputPort0 Subroutine

The InputPort0 subroutine is used to read the input port. After calling the InputPort0 routine, the port value is stored in a working register. Permanent data needs to be saved by the application to a permanent register. In most cases, a permanent register is not needed because the port value can be read using the routine anytime.

```
InputPort0:
      .if     EXTMEMORY                     ; Input Port0
      ld      r0,#^HB(extadd)               ; Point to External I/O
      ld      r1,#^LB(extadd)
      lde     nport0,@rr0                   ; Get Expanded Port0 Data
      ret

      .else
      ld      nport0,port0                  ; Get Real Port0 Data
      ret
      .endif
```

Note the usage of the LDE instruction as opposed to an LDC instruction. The LDE instruction is used to write or read from external memory.

Setting Bit 6 Low

This example shows how to set Bit 6 Low, while not affecting any other bits on the output port.

```
        and     portlsave,#%bf              ; Turn on LED by Setting the Bit Low
        call    OutputPort1
```

Reading in New Data from the Port

This example shows how to read in new data from the port. The data is read into a temporary register to direct a conditional jump instruction. R7 is used for the temporary "nport0" register in the Z89175 Evaluation Kit application code. A regular register also could be used, but a working register was used in the kit code to conserve regular registers for the application.

```
        Call    InputPort0                  ; Get Port0
        tm      nport0,#1                   ; Check if bit0 is High
        jr      nz,PinWasHigh
        .
        .
PinWasHigh:
        .
        .
```

DIGITAL INSTANT WATER HEATER

THE Z8 MCU IS THE IDEAL CHOICE WHEN IMPLEMENTING CONTROL TECHNIQUES FOR DIGITAL INSTANT WATER HEATERS AND OTHER DUTY-CYCLE CONTROL APPLICATIONS.

INTRODUCTION

During the last decade, a broad range of electrical appliances has migrated from simple, analog design to sophisticated digital design. As an example, black and white analog TVs have been replaced by color TVs with on-screen display menu and infrared remote control. Ovens, irons, fans, air-conditioners, rice cookers, and scores of other appliances now use the MCU design instead of conventional, analog designs.

In addition to features that directly benefit the end user, such as LEDs and audio acknowledgment when keys are pressed, the Z8® MCU also benefits the digital design in a number of ways:

- Combined, multiple functions that eliminate many discrete components.
- Enhanced flexibility in changing or adding features to firmware without changing components.
- Reduced cost and improved system reliability.

Heat, light, and speed are the three basic appliance control functions. This application note focuses upon heat control, demonstrating the many advantages of a microcontrolled water heater (using Zilog's Z86C31 device) instead of the conventional, analog water heater. See Figure 1 for a simple functional block diagram.

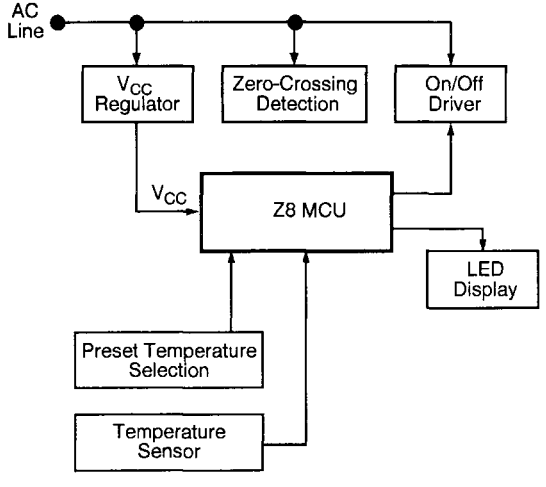


Figure 1. Functional Block Diagram

ADVANTAGES OF THE DIGITAL INSTANT WATER HEATER

Table 1. Conventional Analog versus Digital Instant Water Heater

Analog	Digital
Water is boiled and stored for use.	Boiled water is immediately available. – Save power – Compact design (water tank is not required) – Insulation for hot water is not required
Only "On" or "Off" are available.	Selectable power input by Pulse Width Modulation (PWM)
Hot water is mixed with cold water for desired temperature, requiring two water inlets for adjustment.	Water is boiled to the desired temperature, requiring only one water inlet and one water outlet.
Water temperature changes when water pressure is dropped.	Water temperature is maintained by a power adjustment when water pressure is dropped.
Expensive material, installation, and consumption.	Inexpensive and efficient.
High EMI noise on power line while switching.	Minimal EMI with zero-crossing technology; easily meets EMI regulation requirements.

THEORY OF OPERATION

The conventional method of control is plain and simple: On/Off—full power or no power, and nothing in between. With the Z8® MCU, the power duty cycle can be controlled in a number of ways that vary the percentage of heater power output. The three common duty cycle control methods—Phase Control, Pulse-Width Modulation (PWM), and Zero-Crossing PWM—will each be discussed.

Phase Control

The phase control method conducts power during a specific time period in each half of the 50/60 Hz AC power cycle. Triacs are commonly used to switch on/off the power to loads in AC power systems applications. A gate voltage is required to turn on a triac. Once "On," the triac will generate substantial current to maintain its gate voltage until the AC sine wave approaches zero voltage.

The water power duty cycle can be controlled by simply controlling the fire angle (turn on start time). The triac will deliver the power to load after the fire angle and turn off at the zero-crossing point.

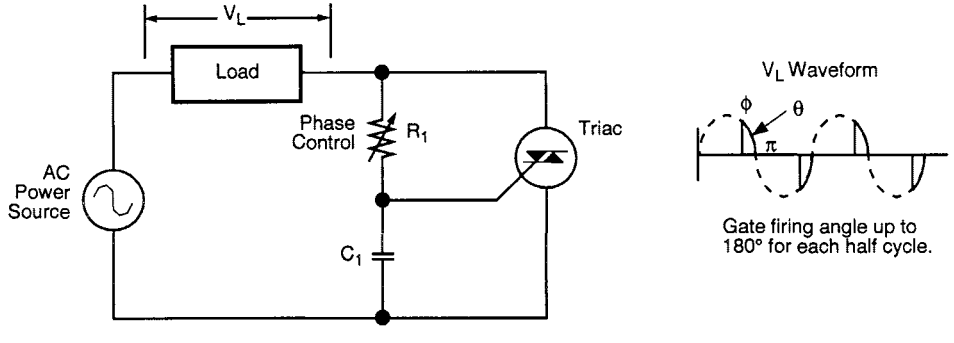


Figure 2. AC Line Phase Control

As shown in Figure 2, the amount of power delivered to the load is equal to the integration of the (V^2 / R) over angle θ. θ is the angle from φ to π.

P = Integration of (V^2/ R) from φ to π.

V = $V_o \sin \theta$

P = Integration of [$V_o^2 \sin^2 \theta$ /R] from φ to π.

= (V_o^2 / R) x { π / 2 - φ /2 + [sin (2 φ)] / 4 }

Full power is when φ = 0, power = (V_o^2 / R) x π / 2.

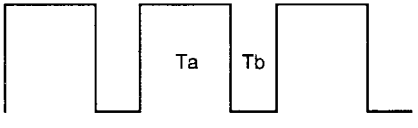
Zero power is when φ = π.

The ratio of power delivery to the full power is { π / 2 - φ/2 + [sin (2 φ)] / 4 } / (π / 2).

Phase control can be easily implemented with a triac and a few passive components. However, the relationship between the firing angle and the power ratio is not linear. Getting an exact proportional output power is somewhat difficult. In addition, switching the triac on at the firing angle may induce high EMI. For a 220V AC line, switching at the peak voltage (311V) will generate a very high EMI noise, as an example. The European Community has imposed some restrictions on the EMI generation of an electrical appliance. Such EMI noise must be suppressed or reduced if the appliance is exported to Europe. As a complementary solution, a snubber or an EMI filter may be added to reduce the noise.

Pulse Width Modulation (PWM)

There are always highs and lows in cyclical signals (see Figure 3). By controlling the width of the highs (Ta) and lows (Tb), the percentage of power (up to full power) delivered to the load can be controlled. PWM is a technique by which the power ratio is controlled by varying the duty cycle of signal high time to total cycle time of the cyclical signal. In this application, the cyclical signal is the AC power line.



$$\text{Frequency} = \frac{1}{Ta + Tb}$$

$$\text{Duty Cycle (Power Ratio)} = \frac{Ta}{Ta + Tb}$$

Figure 3. Pulse Width Modulation

One of the advantages of using PWM is its ease of implementation and subsequent lower cost; however, there are trade-offs to consider. Since the AC waveform is not linear, the power delivered during the peak voltage time will be higher than during the zero-crossing time, which means that power delivery is not linear. Also, the frequency of the PWM signal will affect the smoothness of the power delivered. Generally, higher PWM frequency results in better smoothness. Finally, the On/Off switching of the PWM signal will induce EMI noise, which the designer must minimize.

Zero-Crossing PWM

Since our application uses the AC line at 50/60 Hz, this is the PWM frequency of any PWM power control methodology. Because the power delivery will be relatively unsmooth, the previously discussed PWM technique can be combined with a zero-crossing switching technique to form a "zero-crossing PWM" technique. With AC lines at 50/60 Hz, the zero-crossing PWM technique combines the ease of implementing the PWM methodology while minimizing the EMI noise induced and smoothing the power delivery, which are some of the benefits of the zero-crossing switching technique.

The AC power line is a sine wave with its peak voltage at π / 2 and minimum at 0 or π. If the power line frequency is 60 Hz, there will be 120 (60 Hz x 2) crossings. Power is turned on and off only at the zero-crossing point (see Figure 4). Thus, the EMI will be minimized, and the power applied is smoothed as much as possible.

For example, if delivery of 25 percent of full power is desired, one AC cycle for every four AC cycles of appliance power will be turned on. This means that there are three "Off" AC cycles for every four AC cycles. Since the power potential of each cycle should be the same, there is a 25 percent power ratio. This method is also called "Cycle Skipping."

The other advantages of the zero-crossing PWM technique are accuracy and simplicity in calculating the power ratio. The number of On cycles is counted against the total number of On and Off cycles. A linear relationship exists between the amount of power delivery to the number of On AC cycles. This method also minimizes EMI noise, which is critical for the European market.

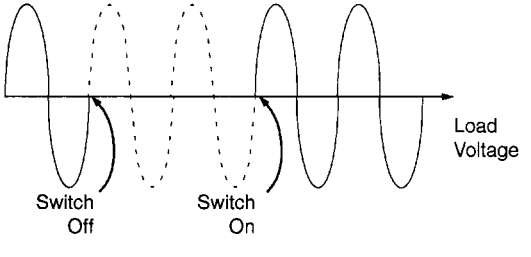


Figure 4. Zero-Crossing Switching

WATER HEATING METHODS

Two common methods are used in electrical water heaters:

1. Constant Power
2. Constant Temperature

Most simple water heaters use the Constant Power method in which constant power is applied, regardless of the water temperature. The Constant Temperature method monitors the water temperature and adjusts the power level so that the temperature is kept in a relatively narrow range. This application uses the Constant Temperature method.

Using a Thermistor

Of the various kinds of temperature sensors—thermo coupler, platinum temperature sensor, and thermistor—the thermistor is used in this application. A thermistor is a device that works by changing resistance values when temperature values change. Thermistors measure water temperature and compare values with preset temperature levels. Water temperature values lower than the preset temperature values cause an output of higher power levels.

Thermistors are frequently used because they are inexpensive and provide an easy interface. Although thermistors are relatively imprecise devices, human sense to temperature is also relatively imprecise; so the combination of these two factors means that our temperature sensing analog-to-digital (A-D) converter with coarse accuracy is acceptable for this application. In particular, the Zilog Z8® MCU comparators provide an economical and efficient way to monitor the water temperature for our application.

User Interface

In contrast to analog sliding switches that have mechanical contacts that wear and corrode with time, the Zilog water heater solution is digital, using push buttons for temperature selection and LEDs to indicate selected water temperature level. The desired output level is selected simply by pressing UP and DOWN buttons.

HARDWARE IMPLEMENTATION

Zero-Crossing Detection

A Z8® MCU controls the power level and senses the water temperature through a thermistor. The Z8 MCU detects the zero-crossing points of the AC power line and applies or removes AC power to the heating elements of the water heater as needed. Refer to Figure 5 for the zero-crossing detection circuitry.

The AC power line is reduced by the potential divider resistors, R1 and R2. Since R2 is 1/100th of R1, the line voltage is divided by 100 times.

V_{IN} = AC Line Voltage x R2 / (R1 + R2)
where $V_{IN} > -0.7V$

Diode D1 is used to clamp the negative input voltage V_{IN} to -0.7V minimum. Too much negative voltage may damage the MCU. The half AC waveform is connected to the positive input of the Z8 comparator (P31).

Prior to zero-crossing detect, port pin P35 of the Z8 is cleared Low so that the reference input of the comparator at P33 is zero volt. When the half AC waveform moving from positive to negative, the comparator output changes from one to zero. The edges of the comparator output signal corresponds to the zero-crossing points of the AC power line. By monitoring this signal, the zero-crossing point can be detected.

International Power Line Considerations

Power line voltages vary from country to country and may be particularly unstable in some developing countries. For example, a peak voltage may drop by as much as 10V when a 15A load is connected. In this case, the zero-crossing switch must be 10 Hz (or higher). The switching frequency can be doubled by switching at every half-cycle. (The power line has two zero crossings at an angle equal to 0 or π.)

DANGER—High Voltage is present. Always use extreme caution when handling the circuit board. PROPER HANDLING PROCEDURES MUST BE USED OR THERE IS DANGER OF ELECTRICAL SHOCK.

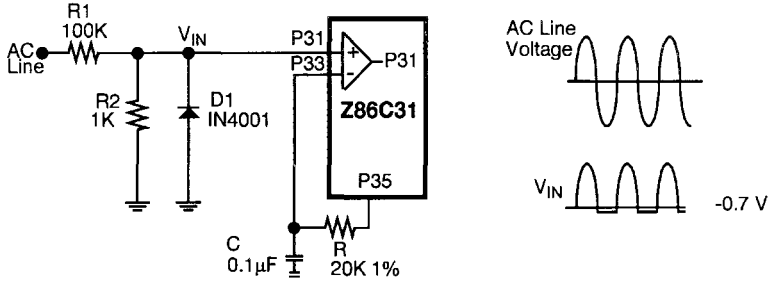


Figure 5. Zero-Crossing Detection

Temperature Detection

The resistance of a thermistor changes with temperature. If current is applied through a thermistor and voltage across it measured against a reference voltage, the temperature can be approximated. The comparators in a Z8® MCU provide an economical way of measuring water temperature with a thermistor.

Figure 6 shows a simple scheme by which the thermistor voltage (V_{CC}-V_T) can be determined as a function of the thermistor resistance R_T and the 6.8K ohm resistor, which is shown.

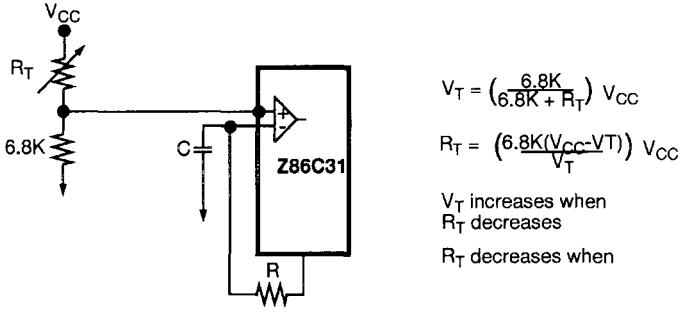


Figure 6. Thermistor Voltage

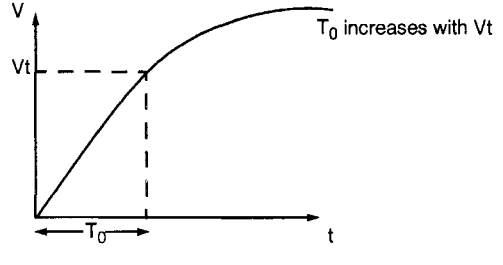
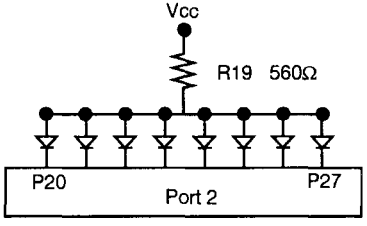


Figure 7. Thermistor Voltage Conversion

User Interface

There are eight levels of selected water temperature. The power-up default is at the preset level 4. The user can select warmer and cooler by pressing the UP and DOWN buttons, respectively. The LEDs indicate the selected level.



LED current = (5V-Vled)/R19
Where Vled is about 1.5V, R19 is 560 Ω
LED current - 6.25 mA

Figure 8. LED Indication

The R4 and R6 10K ohm resistors are used to reduce the Electrostatic Discharge (ESD) current to the Z8® MCU. These two resistors are removable if ESD is not an issue for the product.

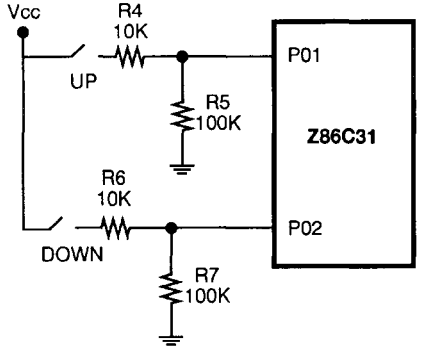


Figure 9. Up and Down Keys

Triac and Triac Driver

A triac is commonly used for duty-cycle control. Similar to a relay, a triac can be On and Off with a relatively small amount of current. Unlike a relay, its response time is much faster and will not generate electric sparks at the moving contact point. An initial current of approximately 50 mA for 40 µS is needed to start it. (Different triacs have different characteristics, so the 50 mA current and 40 µS time are for reference only.) After the triac is On, a self-generated current is essential to maintain the On status. The triac will turn Off automatically when the line voltage is dropped to zero as the self-generated current will also be zero when line voltage is zero.

A triac driver is used to provide a high current to drive the triac. An opto-driver is used if primary and secondary isolation is required. When the Z8 MCU P36 outputs a zero, the internal LED is On, and the triac driver will turn on the triac. With the AC passing the loading (heater coil), the water is heated up to the desired temperature.

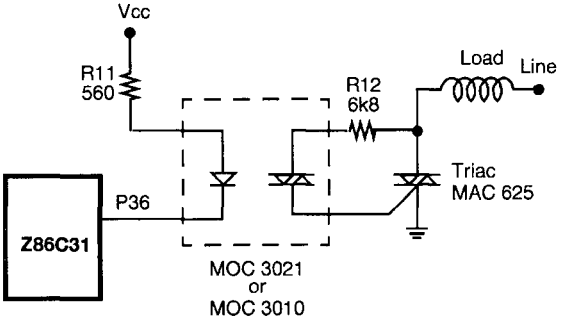


Figure 10. Triac and Triac Driver

DIGITAL INSTANT WATER HEATER SCHEMATIC DIAGRAM

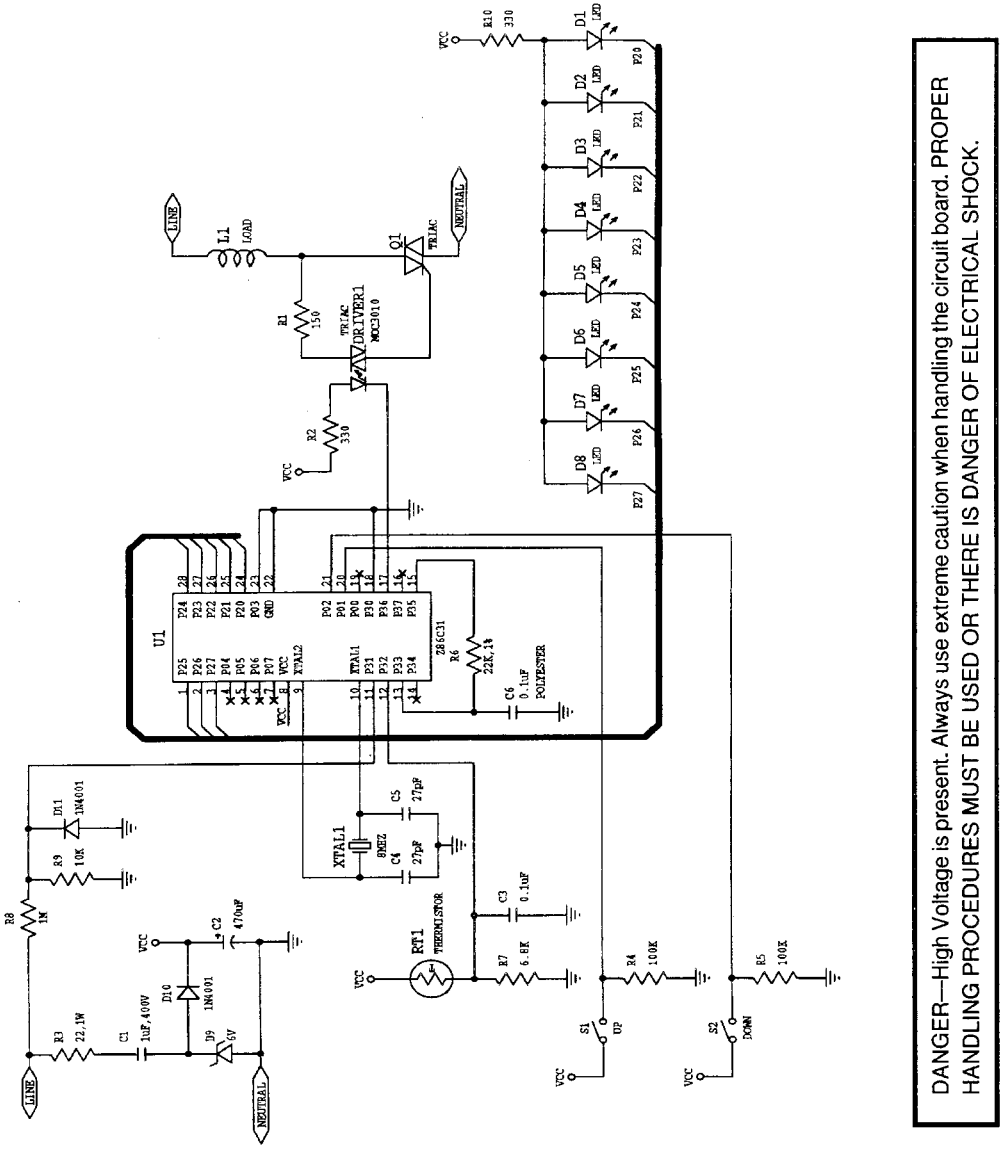


Figure 11. Digital Instant Water Heater Schematic Diagram

SOFTWARE IMPLEMENTAION

Program Operation

The program operation is based on the zero-crossing detection method. The water heater is turned On/Off only at the zero-crossing point. The program will scan the UP and DOWN key once every 8 cycles (160 ms for 50 Hz AC). When any key is detected, it will increment or decrement the selected power level. The Z8® MCU will check the temperature once in every 9 cycles (180 ms for 50 Hz AC).

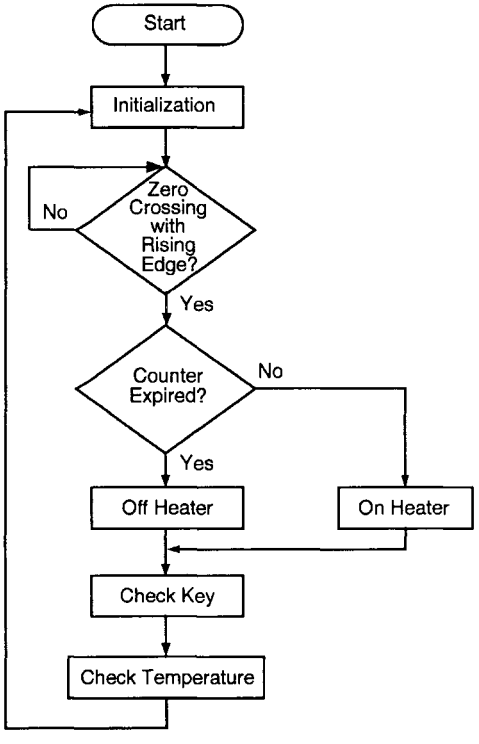


Figure 12. Instant Water Heater Software Flowchart

Temperature Sensing

The Z8® MCU will charge up the capacitor through a 1-percent accuracy resistor. The capacitor voltage rises until it is equal to or more than the thermistor input voltage. The temperature can be estimated by counting the capacitor charging time.

Data Processing

How much to increase or decrease the output power is determined by a comparison of the actual water temperature to the user-selected temperature. The variables are temperature deference, current water temperature, maximum full power output, the current power ratio, and response delay time of the system. For the best temperature control, you may need to "field test" the data processing system.

SUMMARY

The digital instant water heater provides many benefits when compared to conventional, analog solutions:

- User-Friendly Interface
- Precise Water Temperature Control Range
- Minimum EMI Noise
- Closed-Loop Control on the Water Temperature with Minimum Additional Cost

This application note has considered how to implement the zero-crossing technique and the closed-loop feedback temperature control as applied to water heater applications. The same principles, however, also can be applied to other electrical appliances in which duty-cycle control is a consideration.

The Z8® MCU is the ideal controller to implement the control technique—at an affordable cost.

PROGRAM LISTING

Program is assembled by Zilog ZASM assembler.

```
;*********************************************************************
;       FILE    : HEATER.s
        DATE    : 951229
;       MCU     : Z86C31
;       PROJECT : INSTANT WATER HEATER
;*********************************************************************
bitno0          .equ    %01
bitno1          .equ    %02
bitno2          .equ    %04
bitno3          .equ    %08
bitno4          .equ    %10
bitno5          .equ    %20
bitno6          .equ    %40
bitno7          .equ    %80

;*********************************************************************
;       PORTS DEFINITION
;*********************************************************************
;       Port 0 pin
.upkey          .equ    bitno1          ;temp. UP key
.downkey        .equ    bitno2          ;temp. DOWN key
.setp34         .equ    bitno4
.setp37         .equ    bitno7

;       Port 2 pin
;       p20 - p27 : LED
;       0: LED ON
;       1: LED OFF
led0            .equ    11111111b
led1            .equ    11111110b
led2            .equ    11111101b
led3            .equ    11111011b
led4            .equ    11110111b
led5            .equ    11101111b
led6            .equ    11011111b
led7            .equ    10111111b
led8            .equ    01111111b
led9            .equ    00111111b
;       Port 3 pin
;       P31 : AC zero-crossing detection
;       P32 : Temperature sensing
;       P33 : Voltage reference
.charge         .equ    bitno5          ;Charge & discharge cap.
.onheat         .equ    bitno6          ;ON/OFF heater

;*********************************************************************
;       REGISTERS DEFINITION
;*********************************************************************
temp_level      .equ    r1
temp_led        .equ    r2
temp_set        .equ    r3
count           .equ    r4
temp            .equ    r5
power_level     .equ    r6
pwmout          .equ    r7
cold_count      .equ    r8
hot_count       .equ    r12
ledtable        .equ    32
settable        .equ    48
tempA           .equ    64
diff            .equ    7
maxlevel        .equ    16
hot_tc          .equ    22
change2level    .equ    3
changedouble    .equ    6

preset0         .equ    144             ;Temperature preset level 0
preset1         .equ    149             ;Temperature preset level 1
```

```
preset2         .equ    153             ;Temperature preset level 2
preset3         .equ    156             ;Temperature preset level 3
preset4         .equ    163             ;Temperature preset level 4
preset5         .equ    166             ;Temperature preset level 5
preset6         .equ    169             ;Temperature preset level 6
preset7         .equ    172             ;Temperature preset level 7
preset8         .equ    175             ;Temperature preset level 8
preset9         .equ    180             ;Temperature preset level 9

;////////////////////////////////////////////////////////////////////////
;       Macro
;       Refer to technical manuals for Macro definition.
;////////////////////////////////////////////////////////////////////////
bset    .macro  arg1,arg2
      or      arg1,#arg2
      .endm

bclr:   .macro  arg3,arg4
      and     arg3,#^c (arg4) ^& 0ffh
      .endm

brset:      .macro  arg1,arg2,arg3
      tm      arg1,#arg2
      jr      nz,arg3
      .endm

brclr:      .macro  arg1,arg2,arg3
      tm      arg1,#arg2
      jr      z,arg3
      .endm

onheater:   .macro
      bclr    p3,.onheat
      .endm

offheater:  .macro
      bset    p3,.onheat
      .endm
;////////////////////////////////////////////////////////////////////////
            .mlist  on
            .list   on

            .ORG    %0000

;////////////////////////////////////////////////////////
            Interrupt vectors address definition
            No interrupt is used; dummy interrupt return
;////////////////////////////////////////////////////////
            .word   dummy
            .word   dummy
            .word   dummy
            .word   dummy
            .word   dummy
            .word   dummy

;*********************************************
;      PROGRAM STARTS HERE             *
;*********************************************
BEGINNING:  di
            ld      P01M,#00000101b         ;P0-input, internal stack
            clr     P2M                     ;P2 - all output
            ld      P3M,#%02                ;P3 analog
                                            ;P2 open-drain
            clr     SPH
            ld      SPL,#126                ;INIT STACK POINTER
            ld      IPR,#00010011b          ;Set priority of interrupt
            ld      IMR,#%00                ;Set interrupt mask register
```

```
;           INITIALIZE RAM TO ZERO
////////////////////////////////////////
            srp     #%40
            ld      R14,#%3f
ZRAM:       clr     @R14
            djnz    R14,ZRAM
            clr     @R14                    ;Clear register 0

            ld      rp,#%0f                 ;Address extend register %OF
            ld      r0,#11111010b           ;PCON standard mode
                                            ;Port 0 open-drain

            srp     #%10                    ;Set working register
                                            ;Pointer to %10
            ld      count160mS, #8          ;Initialize counter for 160 ms
            ld      count480mS, #9          ;Initialize counter for 480 ms
            ld      hot_count, #hot_tc
            ld      cold_count, #hot_tc
            ld      ledtable,#led0          ;Initialize LED display
            ld      ledtable+1,#led1        ;For different preset levels
            ld      ledtable+2,#led2
            ld      ledtable+3,#led3
            ld      ledtable+4,#led4
            ld      ledtable+5,#led5
            ld      ledtable+6,#led6
            ld      ledtable+7,#led7
            ld      ledtable+8,#led8
            ld      ledtable+9,#led9

            ld      settable,#0             ;Initialize preset temperature level
            ld      settable+1,#preset1
            ld      settable+2,#preset2
            ld      settable+3,#preset3
            ld      settable+4,#preset4
            ld      settable+5,#preset5
            ld      settable+6,#preset6
            ld      settable+7,#preset7
            ld      settable+8,#preset8
            ld      settable+9,#preset9

            ld      temp_level,#4
            ld      power_level,#5
            ld      temp_led,ledtable(temp_level)
            ld      p2,temp_led
            ld      temp_set,settable(temp_level)
            ld      pwmout, power_level

;*******************************************************************************************
;///////////////////////////////////////////////////////////////////////////////////////////
;           Wait for AC Zero-Crossing Input
;           Zero crossing can be negative-to-positive and positive-to-negative.
;           ACcrossing subroutine is only called when AC line is from negative-to-positive.
;           A delay is inserted to filter noise at zero-crossing point.
;///////////////////////////////////////////////////////////////////////////////////////////
ACinput
            brset   p3,bitnol1,ACinput
            call    delay
waitone
            brclr   p3,bitnol,waitone
            call    ACcrossing
            jr      ACinput
;*******************************************************************************************
delay:
            clr     tempA
loop1:
            nop
            nop
            dec     tempA
            jr      nz,loop1
            ret
```

```
;*********************************************************************
////////////////////////////////////////////////
          Check if any key is pressed?
////////////////////////////////////////////////
scankey:
          brset   p0,.upkey,incpreset         ;Up key is pressed?
          brset   p0,.downkey,decpreset       ;Down key is pressed?
          ret

;         Increment key is pressed; increase the preset temperature level.
;         Maximum preset temperature is level 8.
;         Maximum preset power is level 9.

incpreset:
          brset   p0,.upkey,incpreset
          inc     power_level
          inc     temp_level
          cp      temp_level,#8
          jr      c,display
          jr      z,display
          ld      temp_level,#8
          ld      power_level,#9
          jr      display

;         Decrement key is pressed; decrease the preset temperature level.
;         Minimum preset temperature is level 1.
;         Minimum preset power is level 2.

decpreset:
          brset   p0,.downkey,decpreset
          dec     power_level
          dec     temp_level
          jr      nz,display
          ld      temp_level,#1
          ld      power_level,#2
display:
          ld      temp_set,settable(temp_level)
          brclr   p0,bitno3,no_temp_dis
          ld      temp_led,ledtable(temp_level)
          ld      p2,temp_led
no_temp_dis:
          ret

;*************************************************************************
://///////////////////////////////////////////////////////////////////////
;         Analog-to-Digital Conversion on the Thermistor Voltage
;/////////////////////////////////////////////////////////////////////////
ADCtemp:                              ;A-to-D of thermistor voltage
          clr     count
          bset    p3,.charge          ;start charge up C
ADCloop:
          inc     count               ;6 cycles
          jp      z,too_hot           ;6 cycles
          brset   p3,bitno2,ADCloop
;////////////////////////////////////////////////////
;         Total Cycles    = (6+6+10+12)*N
;         Total time      = 1/(8M/2)  * Total Cycles
;                         = 8.5*N uS
;////////////////////////////////////////////////////

;/////////////////////////////////////////////////////////////////////////
;         Compare the preset temperature level and the actual temperature.
;         Change the power level if the temperature difference is too large.
;         Increase change if the difference is larger or lasts longer.
;/////////////////////////////////////////////////////////////////////////
process:
          bclr    p3,.charge          ;Discharge C
          sub     count,temp_set
          jr      c,hotter
check_diff:
          sub     count,#diff
          jr      c,normal_out
```

```
                                    ;Difference higher than #diff
            ld      cold_count,#hot_tc
            dec     hot_count
            jr      nz,normal_o
            ld      hot_count,#1
                                    ;Too hot for 4 sec.
            cp      temp_level,#change2level
            jr      c,normal_o
            jr      z,minus_one
hilevel:
            sub     count,#diff
            jr      c,minus_one
minus_two:
            cp      power_level,#5
            jr      nz,low_1
            ld      pwmout,#2
            jr      end_pwm
low_1:

            ld      pwmout,power_level
            sub     pwmout,#2

            jr      end_pwm
minus_one:
            cp      temp_level,#changedouble
            jr      le,min_one
            jr      minus_two
min_one:
            cp      power_level,#5
            jr      nz,low_3
            inc     count3
            cp      count3,#1
            jr      nz,low_3
            ld      pwmout,#2
            ld      count3,#0
            jr      end_pwm
low_3:

            ld      pwmout,power_level
            cp      power_level,#1
            jr      le,end_pwm
            sub     pwmout,#1
            jr      end_pwm
normal_out:
            ld      hot_count,#hot_tc
            ld      cold_count,#hot_tc
            ld      pwmout,power_level
            jr      end_pwm
normal_o:
            cp      power_level,#5
            jr      nz,low_4
            inc     count3
            cp      count3,#1
            jr      nz,low_4
            ld      pwmout,#2
            ld      count3,#0
            jr      end_pwm
low_4:

            ld      pwmout,temp_level
            cp      pwmout,#1
            jr      le,end_pwm
            sub     pwmout,#1
            jr      end_pwm
hotter:
            ld      hot_count,#hot_tc       ;Reset hot count
            dec     cold_count              ;Decrease cold count
            jr      nz,plusone              ;cold_count = 0 ? (4 sec.)
                                            ;No, goto plusone
            ld      cold_count,#1           ;Yes, load 1 to cold_count
            ld      pwmout,power_level      ;Move temp_level to pwmout
            add     pwmout,#2               ;Add 2 to pwmout
```

```
            cp      pwmout,#9               ;Is pwmout > 9 ?
            jr      ge,pw_1
            jr      end_pwm
pw_1:
            ld      pwmout,#9               ;Yes, load 9 to pwmout
            jr      end_pwm
plusone:
            ld      pwmout,power_level
            add     pwmout,#1
            cp      pwmout,#9
            jr      ge,pw_1

end_pwm:
            brset   p0,bitno3,end_process
            ld      temp_led,ledtable(pwmout)
            ld      p2,temp_led
end_process:
            ret
too_hot:
            ld      count,#255
            jp      process
;////////////////////////////////////////////////////////////////////////
;      AC line change in every 20 ms for 50 Hz AC line.
;      The PWM output is changed at the zero-crossing point.
;      Thermistor voltage is checked every 480 ms.
;////////////////////////////////////////////////////////////////////////

ACcrossing:
outputpwm:
            and     pwmout,pwmout
            jr      nz,pwm_on
            offheater                       ;if pwmout=0 heater=OFF
            jr      check_key
pwm_on:
            dec     pwmout
            onheater
check_key:
            dec     count160mS
            jr      nz,check_ADC
            ld      count160mS,#8                   ;20 ms x 8 = 160 ms
            bclr    p3,.setp34
            bset    p3,.setp37
            call    scankey
            bclr    p3,.setp37
check_ADC:
            dec     count480mS
            jr      nz,ret_AC
            ld      count480mS,#9                   ;20 ms x 24 = 480 ms
            bclr    p3,.setp37
            bset    p3,.setp34
            nop
            call    ADCtemp
            bclr    p3,.setp34
ret_AC:
            ret
;////////////////////////////////////////////////////////////////////////
dummy
            iret
            .end
```

ON-CHIP OSCILLATOR DESIGN

DESIGN AND BUILD RELIABLE, COST-EFFECTIVE, ON-CHIP OSCILLATOR CIRCUITS THAT ARE TROUBLE FREE. PUTTING OSCILLATOR THEORY INTO A PRACTICAL DESIGN MAKES FOR A MORE DEPENDABLE CHIP.

INTRODUCTION

This Application Note (App Note) is written for designers using Zilog Integrated Circuits with on-chip oscillators; circuits in which the amplifier portion of a feedback oscillator is contained on the IC. This App Note covers common theory of oscillators, and requirements of the circuitry (both internal and external to the IC) which comes from the theory for crystal and ceramic resonator based circuits.

Purpose and Benefits

The purposes and benefits of this App Note include:

1. Providing designers with greater understanding of how oscillators work and how to design them to avoid problems.
2. To eliminate field failures and other complications resulting from an unawareness of critical on-chip oscillator design constraints and requirements.

Problem Background

Inadequate understanding of the theory and practice of oscillator circuit design, especially concerning oscillator start-up, has resulted in an unreliable design and subsequent field problems.

OSCILLATOR THEORY OF OPERATION

The circuit under discussion is called the Pierce Oscillator (Figures 1, 2). The configuration used is in all Zilog on-chip oscillators. Advantages of this circuit are low power consumption, low cost, large output signal, low power level in the crystal, stability with respect to V_{CC} and temperature, and low impedances (not disturbed by stray effects). One drawback is the need for high gain in the amplifier to compensate for feedback path losses.

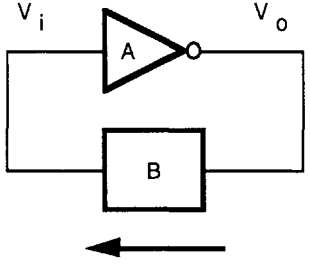


Figure 1. Basic Circuit and Loop Gain

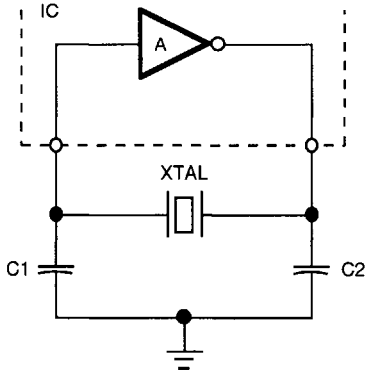


Figure 2. Zilog Pierce Oscillator

OSCILLATOR THEORY OF OPERATION (Continued)

Pierce Oscillator (Feedback Type)

The basic circuit and loop gain is shown in Figure 1. The concept is straightforward; gain of the amplifier is A = Vo/Vi. The gain of the passive feedback element is B = Vi/Vo. Combining these equations gives the equality AB = 1. Therefore, the total gain around the loop is unity. Also, since the gain factors A and B are complex numbers, they have phase characteristics. It is clear that the total phase shift around the loop is forced to zero (i.e., 360 degrees), since V_{IN} must be in phase with itself. In this circuit, the amplifier ideally provides 180 degrees of phase shift (since it is an inverter). Hence, the feedback element is forced to provide the other 180 degrees of phase shift.

Additionally, these gain and phase characteristics of both the amplifier and the feedback element vary with frequency. Thus, the above relationships must apply at the frequency of interest. Also, in this circuit the amplifier is an active element and the feedback element is passive. Thus, by definition, the gain of the amplifier at frequency must be greater than unity, if the loop gain is to be unity.

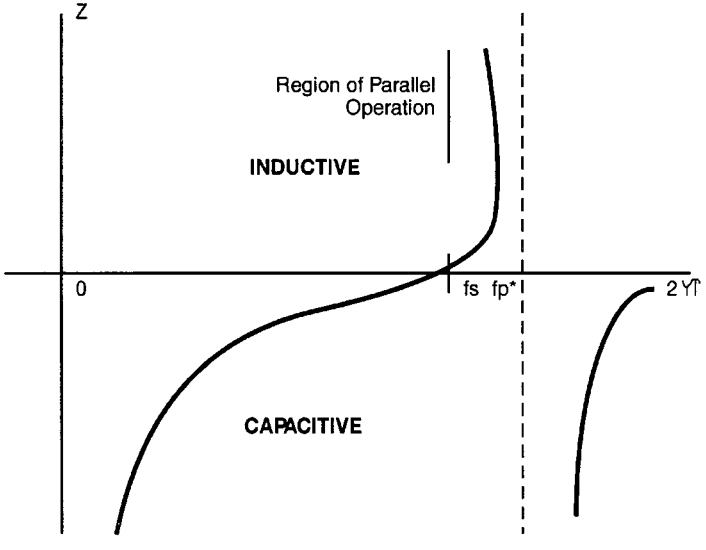
The described oscillator amplifies its own noise at start-up until it settles at the frequency which satisfies the gain/phase requirement AB = 1. This means loop gain equals one, and loop phase equals zero (360 degrees). To do this, the loop gain at points around the frequency of oscillation must be greater than one. This achieves an average loop gain of one at the operating frequency.

The amplifier portion of the oscillator provides gain > 1 plus 180 degrees of phase shift. The feedback element provides the additional 180 degrees of phase shift without attenuating the loop gain to < 1. To do this the feedback element is inductive, i.e., it must have a positive reactance at the frequency of operation. The feedback elements discussed are quartz crystals and ceramic resonators.

Quartz Crystals

A quartz crystal is a piezoelectric device; one which transforms electrical energy to mechanical energy and vice versa. The transformation occurs at the resonant frequency of the crystal. This happens when the applied AC electric field is sympathetic in frequency with the mechanical resonance of the slice of quartz. Since this characteristic can be made very accurate, quartz crystals are normally used where frequency stability is critical. Typical frequency tolerance is .005 to 0.3%.

The advantage of a quartz crystal in this application is its wide range of positive reactance values (i.e., it looks inductive) over a narrow range of frequencies (Figure 3).



* fs - fp is very small (approximately 300 parts per million)

Figure 3. Series vs. Parallel Resonance

However, there are several ranges of frequencies where the reactance is positive; these are the fundamental (desired frequency of operation), and the third and fifth mechanical overtones (approximately 3 and 5 times the fundamental frequency). Since the desired frequency range in this application is always the fundamental, the overtones must be suppressed. This is done by reducing the loop gain at these frequencies. Usually, the amplifier's gain roll off, in combination with the crystal parasitics and load capacitors, is sufficient to reduce gain and prevent oscillation at the overtone frequencies.

The following parameters are for an equivalent circuit of a quartz crystal (Figure 4):

L - motional inductance (typ 120 mH @ 4 MHz)

C - motional capacitance (typ .01 pf @ 4 MHz)

R - motional resistance (typ 36 ohm @ 4 MHz)

Cs - shunt capacitance resulting from the sum of the capacitor formed by the electrodes (with the quartz as a dielectric) and the parasitics of the contact wires and holder (typ 3 pf @ 4 MHz).

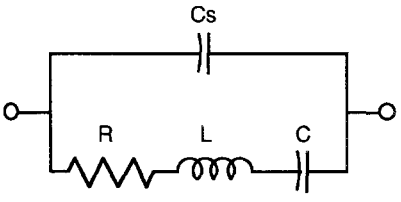
The series resonant frequency is given by:

Fs = 1/(2π x sqrt of LC),
where Xc and Xl are equal.

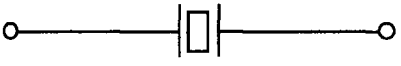
Thus, they cancel each other and the crystal is then R shunted by Cs with zero phase shift.

The parallel resonant frequency is given by:

Fp = 1/[2π x sqrt of L (C Ct/C+Ct)],
where: $Ct = C_L + C_S$



Quartz Equivalent Circuit



Symbolic Representatior

Figure 4. Quartz Oscillator

Series vs. Parallel Resonance. There is very little difference between series and parallel resonance frequencies (Figure 3). A series resonant crystal (operating at zero phase shift) is desired for non-inverting amplifiers. A parallel resonant crystal (operating at or near 180 degrees of phase shift) is desired for inverting amps. Figure 3 shows that the difference between these two operating modes is small. Actually, all crystals have operating points in both serial and parallel modes. A series resonant circuit will NOT have load caps C1 and C2. A data sheet for a crystal designed for series operation does not have a load cap spec. A parallel resonant crystal data sheet specifies a load cap value which is the series combination of C1 and C2. For this App Note discussion, since all the circuits of interest are inverting amplifier based, only the parallel mode of operation is considered.

OSCILLATOR THEORY OF OPERATION

Ceramic Resonators

Ceramic resonators are similar to quartz crystals, but are used where frequency stability is less critical and low cost is desired. They operate on the same basic principle as quartz crystals as they are piezoelectric devices and have a similar equivalent circuit. The frequency tolerance is wider (0.3 to 3%), but the ceramic costs less than quartz.

Figure 5 shows reactance vs. frequency and Figure 6 shows the equivalent circuit.

Typical values of parameters are L = .092 mH, C = 4.6 pf, R = 7 ohms and Cs = 40 pf, all at 8 MHz. Generally, ceramic resonators tend to start up faster but have looser frequency tolerance than quartz. This means that external circuit parameters are more critical with resonators.

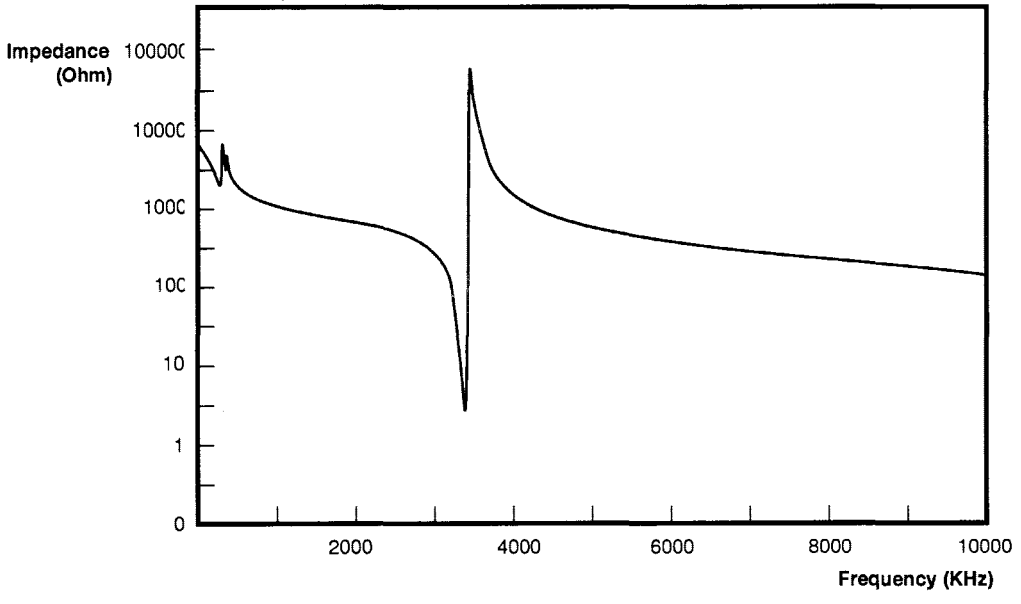


Figure 5. Ceramic Resonator Reactance

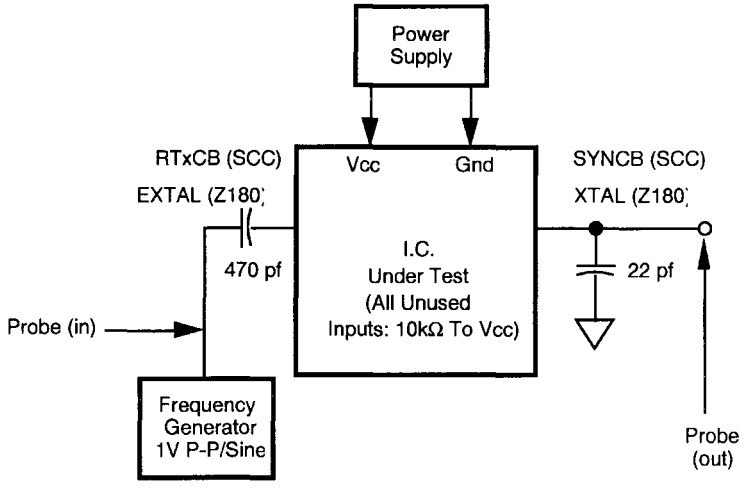


Figure 6. Gain Measurement

Load Capacitors

The effects/purposes of the load caps are:

Cap C2 combined with the amp output resistance provides a small phase shift. It also provides some attenuation of overtones.

Cap C1 combined with the crystal resistance provides additional phase shift.

These two phase shifts place the crystal in the parallel resonant region of Figure 3.

Crystal manufacturers specify a load capacitance number. This number is the load seen by the crystal which is the series combination of C1 and C2, including all parasitics (PCB and holder). This load is specified for crystals meant to be used in a parallel resonant configuration. The effect on start-up time; if C1 and C2 increase, start-up time increases to the point at which the oscillator will not start. Hence, for fast and reliable start-up, over manufacture of large quantities, the load caps should be sized as low as possible without resulting in overtone operation.

Amplifier Characteristics

The following text discusses open loop gain vs. frequency, open loop phase vs. frequency, and internal bias.

Open Loop Gain vs. Frequency over Iot, VCC, Process Split, and Temp. Closed loop gain must be adequate to start the oscillator and keep it running at the desired frequency. This means that the amplifier open loop gain must be equal to one plus the gain required to overcome the losses in the feedback path, across the frequency band and up to the frequency of operation. This is over full process, Iot, V_{CC}, and temperature ranges. Therefore, measuring the open loop gain is not sufficient; the losses in the feedback path (crystal and load caps) must be factored in.

Open Loop Phase vs. Frequency. Amplifier phase shift at and near the frequency of interest must be 180 degrees plus some, minus zero. The parallel configuration allows for some phase delay in the amplifier. The crystal adjusts to this by moving slightly down the reactance curve (Figure 3).

Internal Bias. Internal to the IC, there is a resistor placed from output to input of the amplifier. The purpose of this feedback is to bias the amplifier in its linear region and to provide the start-up transition. Typical values are 1M to 20M ohms.

PRACTICE: CIRCUIT ELEMENT AND LAY OUT CONSIDERATIONS

The discussion now applies prior theory to the practical application.

Amplifier and Feedback Resistor

The elements of the circuit, internal to the IC, include the amplifier, feedback resistor, and output resistance. The amplifier is modeled as a transconductance amplifier with a gain specified as I_{OUT}/V_{IN} (amps per volt).

Transconductance/Gain. The loop gain AB = gm x Z1, where gm is amplifier transconductance (gain) in amps/volt and Z1 is the load seen by the output. AB must be greater than unity at and about the frequency of operation to sustain oscillation.

Gain Measurement Circuit. The gain of the amplifier can be measured using the circuits of Figures 6 & 7. This may be necessary to verify adequate gain at the frequency of interest and in determining design margin.

Gain Requirement vs. Temperature, Frequency and Supply Voltage. The gain to start and sustain oscillation (Figure 8) must comply with:

$$gm > 4\pi^2 f^2 Rq\ C_{IN}\ C_{OUT}t \times M$$

where: M is a quartz form factor = $(1 + C_{OUT}/C_{IN} + C_{OUT}/C_{OUT})^2$

Output Impedance. The output impedance limits power to the XTAL and provides small phase shift with load cap C2.

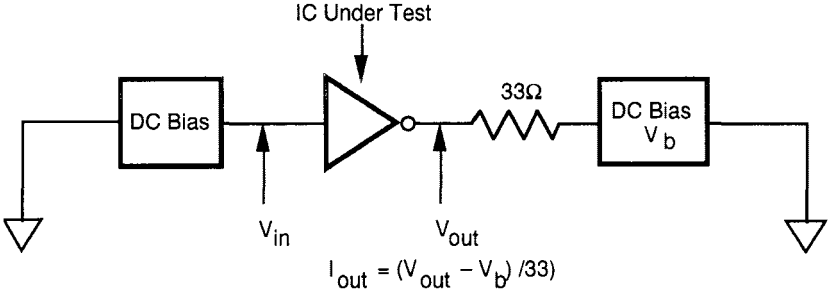
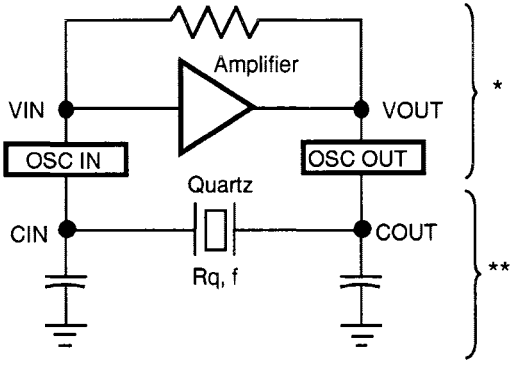


Figure 7. Transconductance (gm) Measurement



* Inside chip, feedback resistor biases the amplifier in the high gm region.

** External components typically: CIN = COUT = 30 to 50 pf (add 10 pf pin cap).

Figure 8. Quartz Oscillator Configuration

Load Capacitors

In the selection of load caps it is understood that parasitics are always included.

Upper Limits. If the load caps are too large, the oscillator will not start because the loop gain is too low at the operating frequency. This is due to the impedance of the load capacitors. Larger load caps produce a longer start-up.

Lower Limits. If the load caps are too small, either the oscillator will not start (due to inadequate phase shift around the loop), or it will run at a 3rd, 5th, or 7th overtone frequency (due to inadequate suppression of higher overtones).

Capacitor Type and Tolerance. Ceramic caps of ±10% tolerance should be adequate for most applications.

Ceramic vs. Quartz. Manufacturers of ceramic resonators generally specify larger load cap values than quartz crystals. Quartz C is typically 15 to 30 pf and ceramic typically 100 pf.

Summary. For reliable and fast start-up, capacitors should be as small as possible without resulting in overtone operation. The selection of these capacitors is critical and all of the factors covered in this note should be considered.

Feedback Element

The following text describes the specific parameters of a typical crystal:

Drive Level. There is no problem at frequencies greater than 1 MHz and V_{CC} = 5V since high frequency AT cut crystals are designed for relatively high drive levels (5-10 mw max).

A typical calculation for the approximate power dissipated in a crystal is:

$$P = 2R (\pi \times f \times C \times V_{CC})^2$$

Where. R = crystal resistance of 40 ohms, C = C1 + Co = 20 pf. The calculation gives a power dissipation of 2 mW at 16 MHz.

Series Resistance. Lower series resistance gives better performance but costs more. Higher R results in more power dissipation and longer start-up, but can be compensated by reduced C1 and C2. This value ranges from 200 ohms at 1 MHz down to 15 ohms at 20 MHz.

Frequency. The frequency of oscillation in parallel resonant circuits is mostly determined by the crystal (99.5%). The external components have a negligible effect (0.5%) on frequency. The external components (C1,C2) and layout are chosen primarily for good start-up and reliability reasons.

Frequency Tolerance (initial temperature and aging). Initial tolerance is typically ±.01%. Temperature tolerance is typically ±.005% over the temp range (-30 to +100 degrees C). Aging tolerance is also given, typically ±.005%.

Holder. Typical holder part numbers are HC6, 18, 25, 33, 44.

Shunt Capacitance. (Cs) typically <7 pf.

Mode. Typically the mode (fundamental, 3rd or 5th overtone) is specified as well as the loading configuration (series vs. parallel).

The ceramic resonator equivalent circuit is the same as shown in Figure 4. The values differ from those specified in the theory section. Note that the ratio of L/C is much lower than with quartz crystals. This gives a lower Q which allows a faster start-up and looser frequency tolerance (typically ±0.9% over time and temperature) than quartz.

Layout

The following text explains trace layout as it affects the various stray capacitance parameters (Figure 9).

Traces and Placement. Traces connecting crystal, caps, and the IC oscillator pins should be as short and wide as possible (this helps reduce parasitic inductance and resistance). Therefore, the components (caps and crystal) should be placed as close to the oscillator pins of the IC as possible.

Grounding/Guarding. The traces from the oscillator pins of the IC should be guarded from all other traces (clock, V_{CC}, address/data lines) to reduce crosstalk. This is usually accomplished by keeping other traces away from the oscillator circuit and by placing a ground ring around the traces/components (Figure 9).

Measurement and Observation

Connection of a scope to either of the circuit nodes is likely to affect operation because the scope adds 3-30 pf of capacitance and 1M-10M ohms of resistance to the circuit.

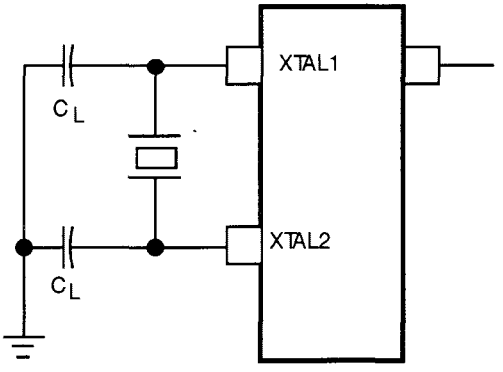
PRACTICE: CIRCUIT ELEMENT AND LAYOUT CONSIDERATIONS (Continued)

Indications of an Unreliable Design

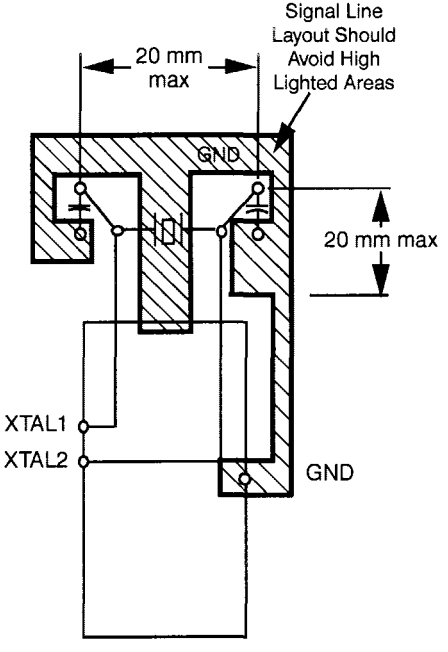
There are two major indicators which are used in working designs to determine their reliability over full lot and temperature variations. They are:

Start Up Time. If start up time is excessive, or varies widely from unit to unit, there is probably a gain problem. C1/C2 needs to be reduced; the amplifier gain is not adequate at frequency, or crystal Rs is too large.

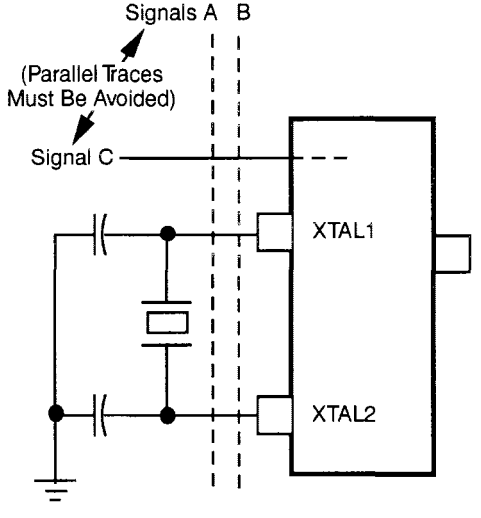
Output Level. The signal at the amplifier output should swing from ground to V_{CC}. This indicates there is adequate gain in the amplifier. As the oscillator starts up, the signal amplitude grows until clipping occurs, at which point, the loop gain is effectively reduced to unity and constant oscillation is achieved. A signal of less than 2.5 Vp-p is an indication that low gain may be a problem. Either C1/C2 should be made smaller or a low R crystal should be used.



Clock Generator Circuit



Board Design Example
(Top View)



- To prevent induced noice, the crystal and load capacitors should be physically located as close to the LSI as possible.
- Signal lines should not run parallel to the clock oscillator inputs.
- V_{CC} power lines should be separated from the clock oscillator input circuitry.

Figure 9. Circuit Board Design Rules

SUMMARY

Understanding the Theory of Operation of oscillators, combined with practical applications, should give designers enough information to design reliable oscillator circuits. Proper selection of crystals and load capacitors, along with good layout practices, results in a cost effective, trouble free design. Reference the following text for Zilog products with on-chip oscillators and their general/specific requirements.

ZILOG PRODUCT USING ON-CHIP OSCILLATORS

Zilog products that have on-chip oscillators:

Z8® Family: All
Z80®: C01, C11, C13, C15, C50, C90, 180, 181, 280
Z8000®: 8581
Communications Products: SCC™, ISCC™, ESCC™

ZILOG CHIP PARAMETERS

The following are some recommendations on values/parameters of components for use with Zilog on-chip oscillators. These are only recommendations; no guarantees are made by performance of components outside of Zilog ICs. Finally, the values/parameters chosen depend on the application. This App Note is meant as a guideline to making these decisions. Selection of optimal components is always a function of desired cost/performance tradeoffs.

Note: All load capacitance specs include stray capacitance.

Z8® Family

General Requirements: 8 to 20 MHz

Crystal Cut: AT cut, parallel resonant, fundamental mode.
Crystal Co: < 7 pf for all frequencies.
Crystal Rs: < 100 ohms for all frequencies.
Load Capacitance: 10 to 22 pf, 15 pf typical.

Z8000® Family (8581 only)

General Requirements:

Crystal cut: AT cut, parallel resonant, fundamental mode.
Crystal Co: < 7 pf for all frequencies.
Crystal Rs: < 150 ohms for all frequencies.
Load capacitance: 10 to 33 pf.

Z80® Family

General Requirements:

Crystal cut: AT cut, parallel resonant, fundamental mode.
Crystal Co: < 7 pf for all frequencies.
Crystal Rs: < 60 ohms for all frequencies.
Load capacitance: 10 to 22 pf.

Communications Family

General Requirements:

Crystal cut: AT cut, parallel resonant, fundamental mode.
Crystal Co: < 7 pF for all frequencies.
Crystal Rs: < 150 ohms for all frequencies.
Load capacitance: 20 to 33 pF.
Frequency: cannot exceed PCLK.

REFERENCES MATERIALS AND ACKNOWLEDGMENTS

Intel Corp., Application Note AP-155, "Oscillators for Micro Controllers", order #230659-001, by Tom Williamson, Dec. 1986.

Motorola 68HC11 Reference Manual.

National Semiconductor Corp., App Notes 326 and 400.

Zilog, Inc., Steve German; Figures 4 and 8.

Zilog, Inc., Application Note, "Design Considerations Using Quartz Crystals with Zilog Components" - Oct. 1988.

Data Sheets; CTS Corp. Knights Div., Crystal Oscillators.

Retro Workshop 果粉工作室

THE Z86C08 CONTROLS A SCROLLING LED MESSAGE DISPLAY

DISPLAY TEXT AND GRAPHICS WHILE SCROLLING THE LED MESSAGE WITH A MINIMUM OF HARDWARE. THE CHARACTERS ARE DISPLAYED USING A TIME-DIVISION MULTIPLEX TECHNIQUE WITH MORE THAN SIX CHARACTERS EASILY ADDED BY SOFTWARE.

INTRODUCTION

Designed around the Zilog Z86C08, 18-pin microcontroller, this LED display is capable of displaying text as well as graphics, with hardware being kept to a minimum. The present design is configured to display u to six characters, but additional characters are easily added with minimal software changes.

The display uses a TDM (Time-Division Multiplexed) technique to display the characters. That is, each character segment is turned on for a few hundred microseconds at a time, then is "refreshed" every 18 ms.

For demonstration purposes, the software routine displays the time in hours, minutes, and seconds. Once every sixty seconds, a "secret message" scrolls across the display. Then, after the message is displayed, the program reverts back to displaying the time.

THE HARDWARE

The Z86C08 (Figure 1) uses Port 2 for the row data and clocks the column data out of Port 0. PNP transistors are used to drive the rows, since the Z86C08 cannot source the required current directly. The characters are displayed in a 5x7 format, so only seven lines are needed out of Port 2. A logic Low turns on the transistors, while a logic High turns them off.

THE HARDWARE (Continued)

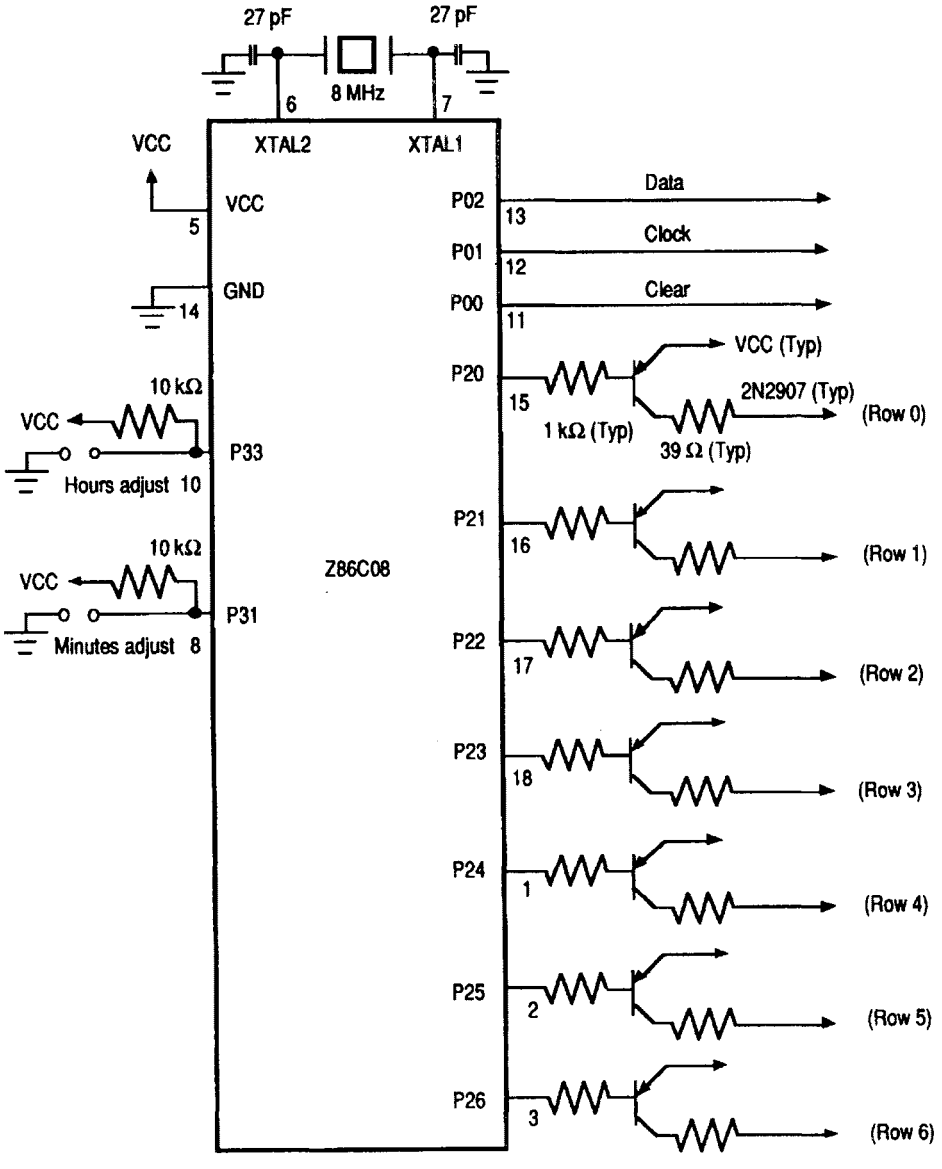


Figure 1. Z86C08 Circuit Functions

The columns are driven by six 74HCT164 shift registers (Figure 2). The 74HCT164s do not have the necessary sink capability to drive the LEDs, so the outputs of the shift registers drive six 75492 high-current buffers. These are capable of sinking 250 mA per pin continuously, with instantaneous current per column approaching 1.5 A. Each 74HCT164 drives six columns; five character segments, plus one space. The last Q output from the shift register is then fed to the DATA input of the next shift register, while all CLEAR and CLOCK lines are wired in parallel.

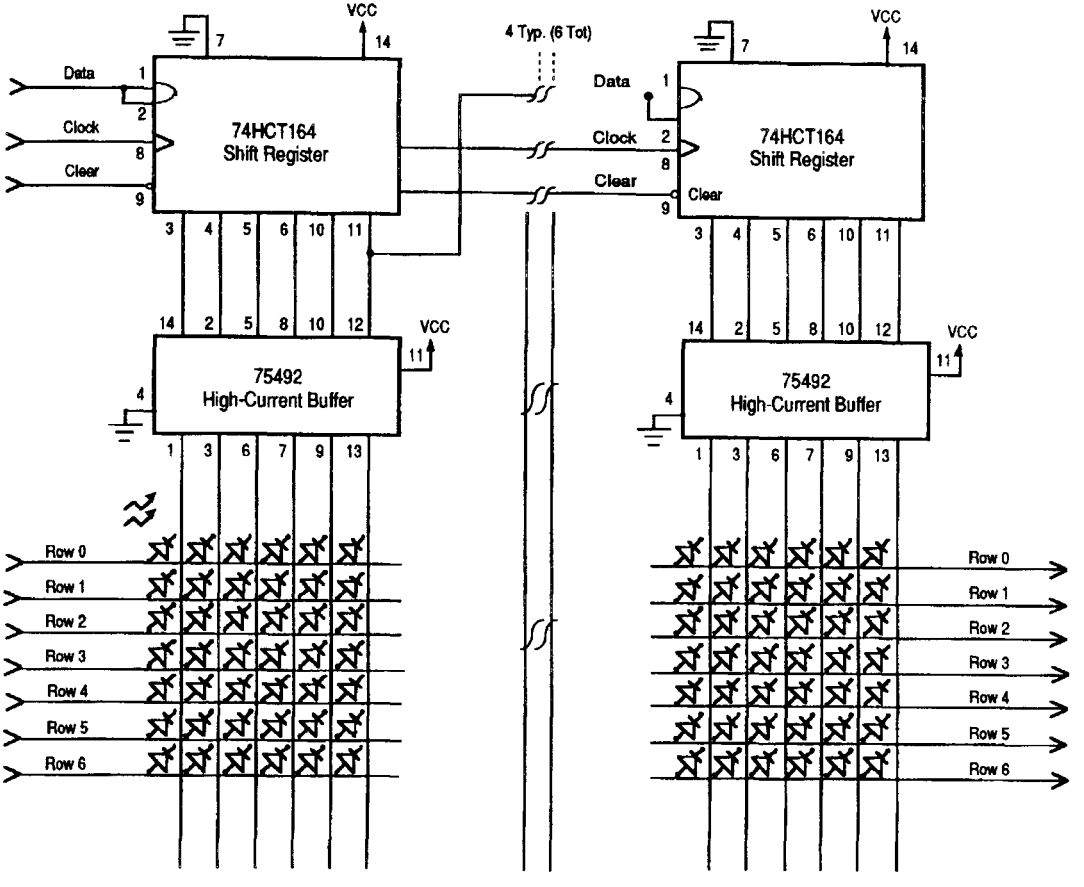


Figure 2. LED Circuitry

THE HARDWARE (Continued)

To scan the display, a logic 1 is output at P0-2. Next, P0-1 is taken High, then immediately taken Low, along with P0-2. This clocks a logic 1 into the first shift register (column 1). Next a character segment is output at P2. The transistors that are turned on will source current for the LEDs with the column providing a sink path. The columns are left on for about 500 µs and then turned off. Now, the column data is shifted one and a new character segment is output. After the last column has been scanned, the display is "refreshed" starting at the first column again.

To set the time, two push-button switches are connected to Port 3 to adjust the hours and minutes.

THE SOFTWARE

The initialization part of the program configures the ports, timers, interrupts, etc. The size of the display buffer (FIFO) is determined by the number of columns in the display. The bottom of the display buffer starts at 20H. The upper limit of the display buffer can extend to 70H. This translates into a sixteen-character display. There is no need to have a display buffer larger than the display itself, since only that many characters can be viewed at a time. A power-up, the display is configured for showing the time.

The flowchart for the display appears in Figures 3a and 3b. To keep time, the internal clock was divided down by T0 to provide an interrupt every 5 ms. The interrupt routine increments a counter, and when 200 counts is reached, the seconds register is incremented by one. Also, when ten seconds is reached, tens-of-seconds is incremented. This counter continues to increment minutes, tens-of-minutes, hours, and tens-of-hours. Upon power-up, the display shows 12:00. When it is around 9:00, the leading hours digit is blanked. Time is adjusted by two push-button switches. When pressed, one increments the minutes register every second, while the other increments the hours register every second. The time data is stored in locations 09H-0EH.

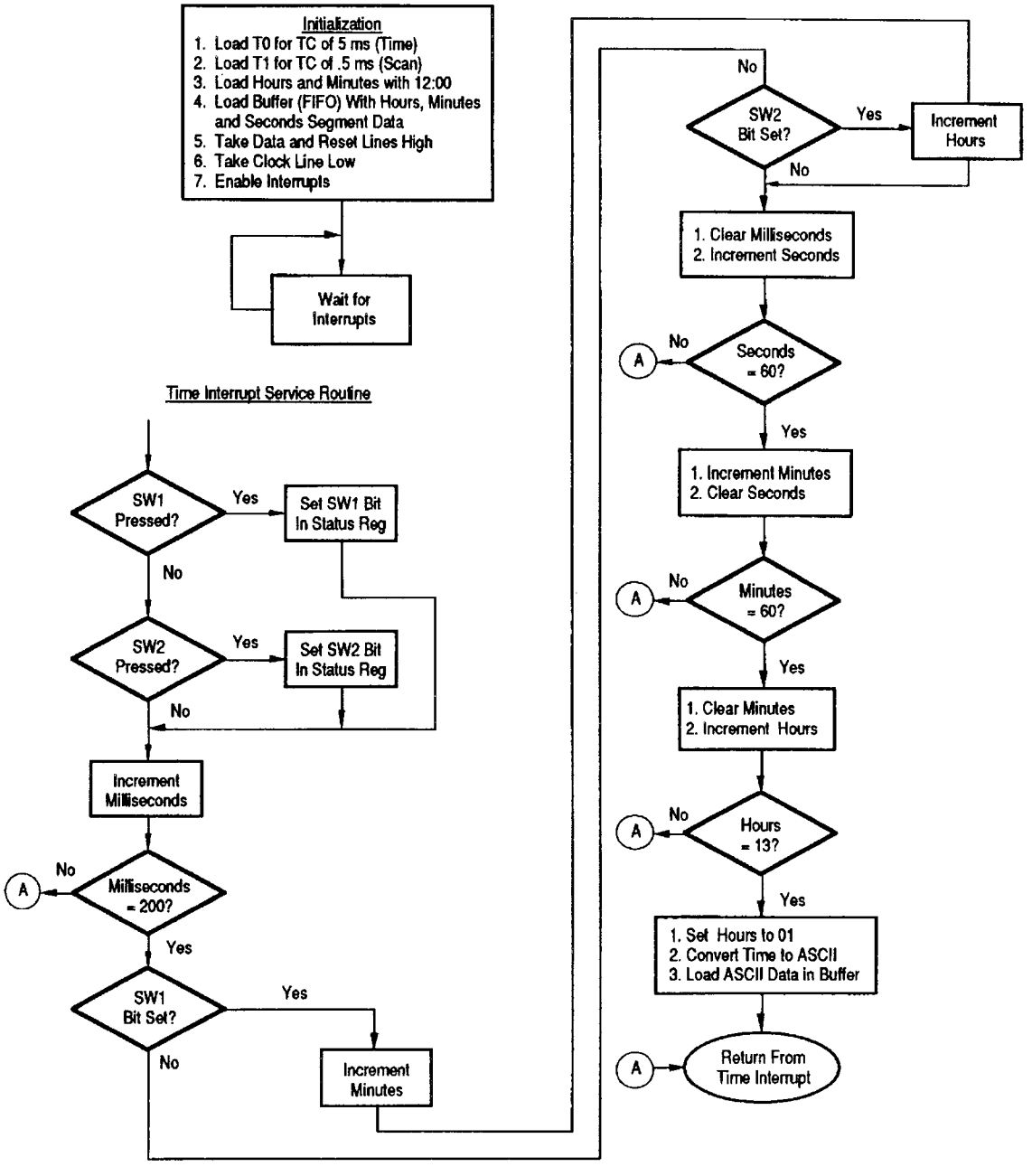


Figure 3a. Display Flowchart

THE SOFTWARE (Continued)

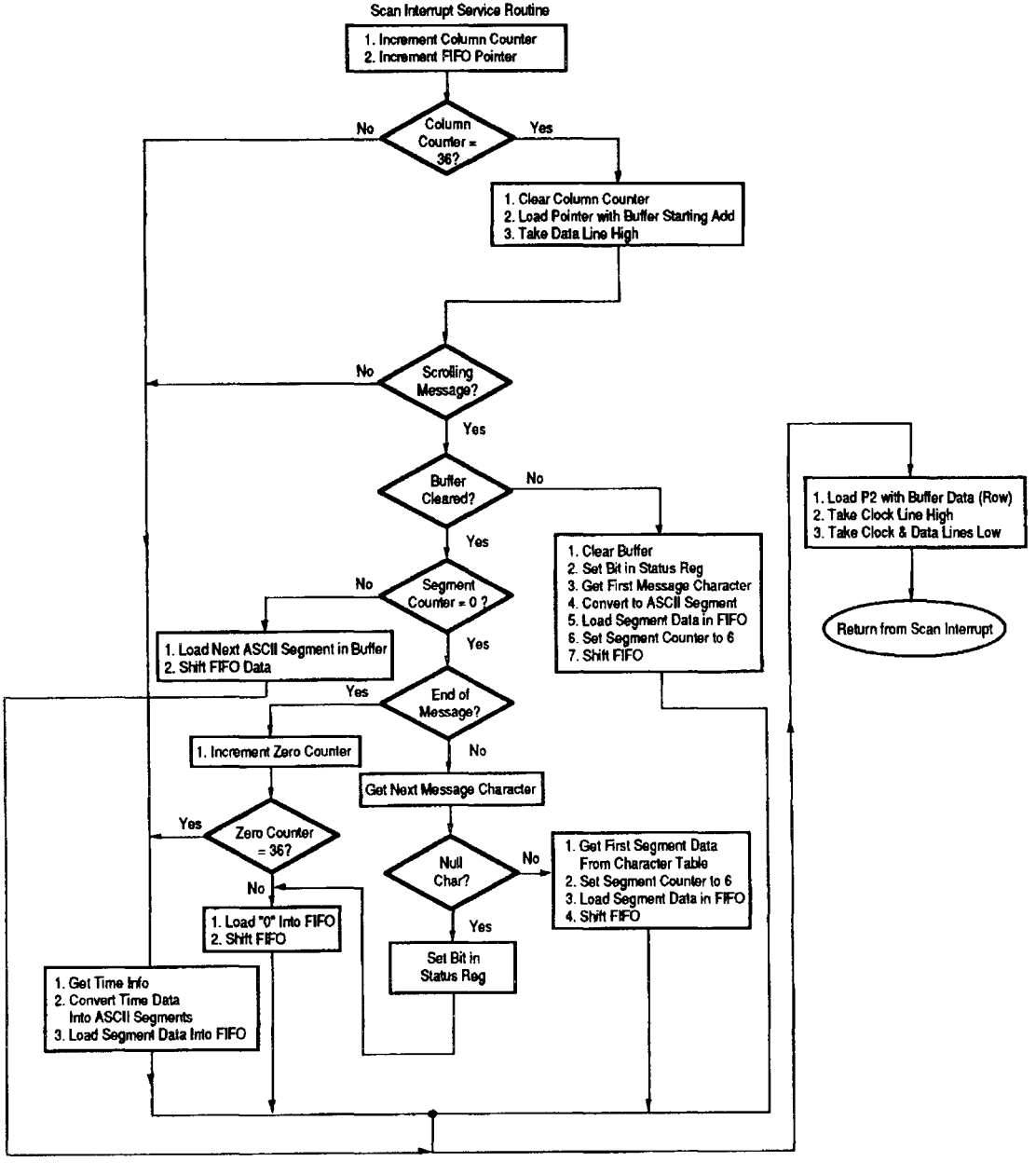


Figure 3b. Display Flowchart

At sixty-second intervals, the time display is blanked, and the internal message is scrolled across the display. The message is stored in an ASCII format. The individual ASCII characters index a look-up table, which converts the characters to a 6x7 format (first segment is a space). The software checks to see if all of the segments have been indexed at the beginning of the scan. If so, it then indexes the next character (Figures 4a and 4b).

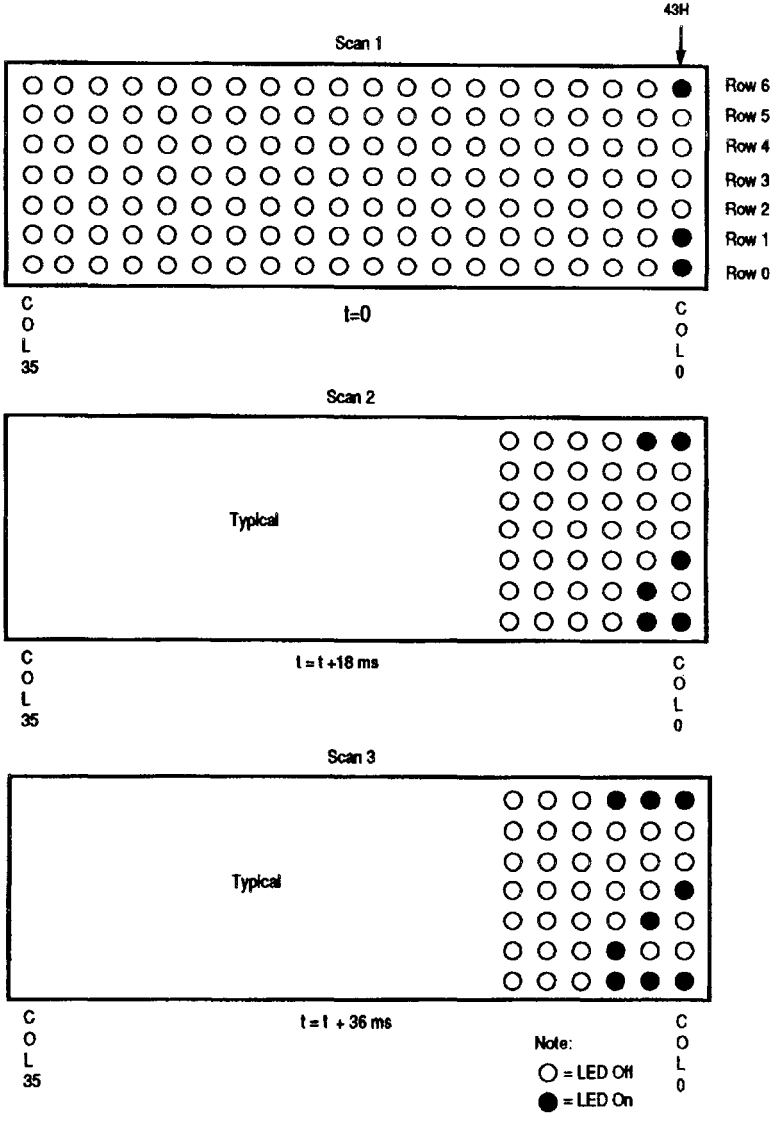


Figure 4a. Scrolling the Letter Z.

For scrolling messages, the display buffer acts like a FIFO (First In - First Out). The FIFO is cleared at power-up. When the internal message is being indexed, the character segments are queued up in the FIFO. The FIFO size is determined by the size of the display. At the end of each scan, the next character segment is indexed, and is stored at the bottom of the FIFO. The character segments are then shifted up the FIFO one location. This process continues until the entire message is displayed. At this time, a 0 is loaded into the FIFO at the beginning of each scan allowing the columns trailing the message to blank out. As the display is being scanned, the byte at each FIFO location is output at P2 as each column is turned on. The scrolling effect is created by shifting the FIFO data at the start of each scan (Figure 5).

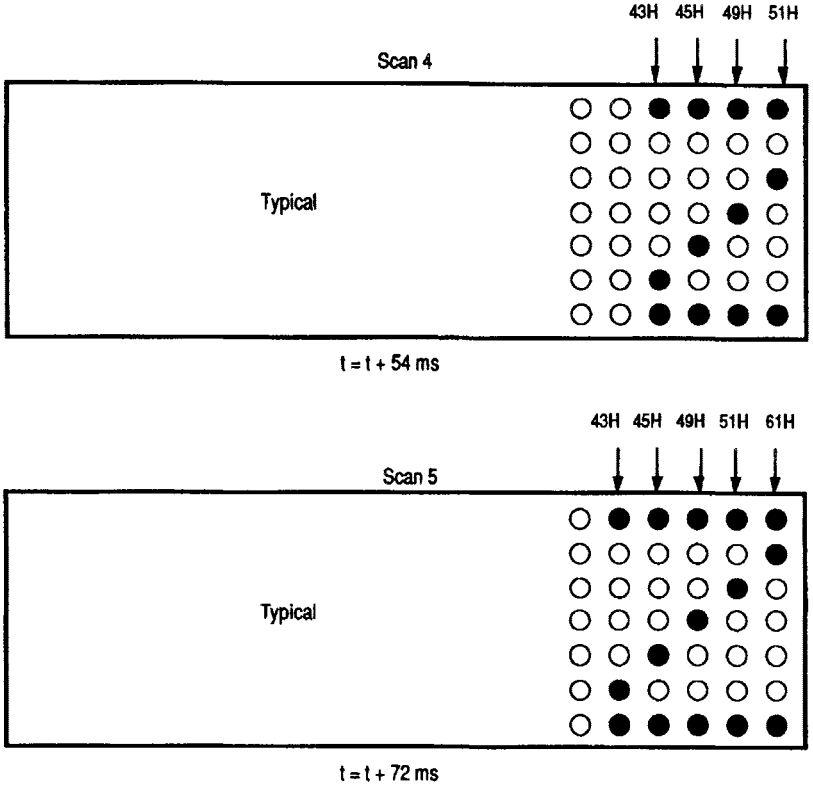


Figure 4b. Scrolling the Letter Z.

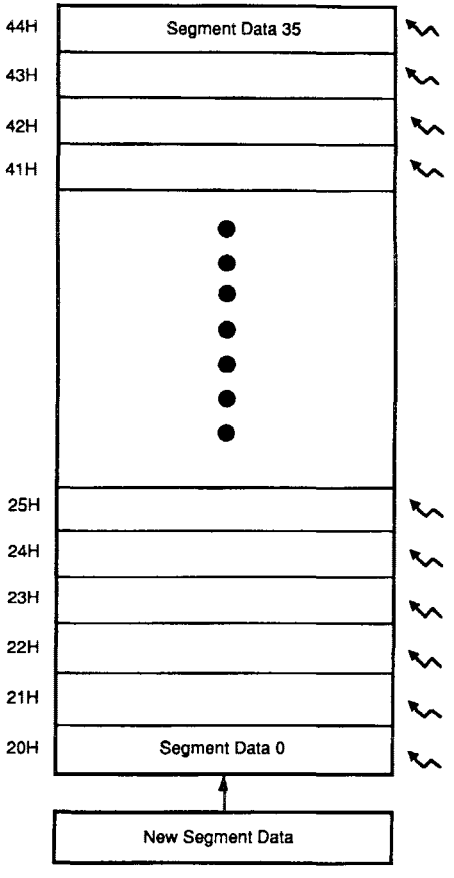


Figure 5. Shifting FIFO Data

APPENDIX

```
                        SCROLL1
                    LINE# --- SOURCE ---
                       1 ;*****************************************************************************;
                       2 ;                                                                             ;
                       3 ;   This is a scrolling LED display routine using the Zilog Z86C08, 'C17     ;
                       4 ;  18-pin CMOS processor. The processor's P2 port outputs the row data, and  ;
                       5 ;  the column data is clocked out of P0 into cascaded shift registers. It     ;
                       6 ;  has a real-time clock in software, and displays the time in hours, min-    ;
                       7 ;  utes, and seconds. At every 60-second interval, it blanks the screen and  ;
                       8 ;  displays an internal message. The message is stored in ROM, and can be up ;
                       9 ;  to 127 characters in length. After scrolling the message, the program re-  ;
                      10 ;  sumes the time display, until the next 60 seconds.                         ;
                      11 ;   The size of the display is 36-columns long, enough to display six char-   ;
                      12 ;  acters. The display can be made longer, but the refresh time may need ad-  ;
                      13 ;  justing to eliminate flicker. Current refresh time is 500 microseconds,   ;
                      14 ;  using a crystal frequency of 8.00 MHz.                                    ;
                      15 ;   There are two pushbutton switches that are read to adjust the hours and  ;
                      16 ;  minutes. On power-up, the display will show 12:00.                         ;
                      17 ;   This program was written by Don Owen Newquist on 12-1-91.                ;
                      18 ;*****************************************************************************;
0000000000000000010   19 WORK_REG       .equ   10h
  abs 00000000        20 address_hi      .equ   r0
  abs 00000001        21 address_lo      .equ   r1
  abs 00000000        22 address         .equ   rr0
  abs 00000002        23 pointer         .equ   r2
  abs 00000004        24 zero_count      .equ   r4
  abs 00000005        25 temp_1          .equ   r5
  abs 00000006        26 temp_2          .equ   r6
  abs 00000007        27 temp_3          .equ   r7
  abs 00000008        28 col_count       .equ   r8
  abs 00000009        29 seq_count       .equ   r9
0000000000000000000   30 TIME_REG        .equ   00h
  abs 00000005        31 millisec        .equ   r5
  abs 00000006        32 seconds         .equ   r6
  abs 00000007        33 minutes         .equ   r7
  abs 00000008        34 hours           .equ   r8
  abs 00000009        35 seconds_lo      .equ   r9
  abs 0000000a        36 seconds_hi      .equ   r10
  abs 0000000b        37 minutes_lo      .equ   r11
  abs 0000000c        38 minutes_hi      .equ   r12
  abs 0000000d        39 hours_lo        .equ   r13
  abs 0000000e        40 hours_hi        .equ   r14
  abs 0000000f        41 sw_count        .equ   r15
0000000000000000004   42 STATUS          .equ   %04
0000000000000000005   43 MILLISEC        .equ   %05
0000000000000000006   44 SECONDS         .equ   %06
0000000000000000007   45 MINUTES         .equ   %07
0000000000000000008   46 HOURS           .equ   %08
0000000000000000009   47 SECONDS_LO      .equ   %09
000000000000000000a   48 SECONDS_HI      .equ   %0a
000000000000000000b   49 MINUTES_LO      .equ   %0b
000000000000000000c   50 MINUTES_HI      .equ   %0c
000000000000000000d   51 HOURS_LO        .equ   %0d
000000000000000000e   52 HOURS_HI        .equ   %0e
0000000000000000020   53 BUFFER          .equ   %20
                      54 ;
                      55 ;               STATUS REG:    d4 d3 d2 d1 d0
                      56 ;                                |  |  |  |  |_ displaying message
                      57 ;                                |  |  |  |___ buffer cleared
```

```
                        58 ;                                  |  |  |_____ end of message
                        59 ;                                  |  |________ min sw pressed
                        60 ;                                  |___________ hrs sw pressed
                        61 ;
00000000                62                .org    00
00000000 0000           63                .word   00
00000002 0000           64                .word   00
00000004 0000           65                .word   00
00000006 0000           66                .word   00
00000008 Wwww           67                .word   time
0000000a Wwww           68                .word   shift
0000000c                69                .org    000ch
                        70 ;*******************************************************************************;
                        71 ;                Initialization                                                 ;
                        72 ;*******************************************************************************;
0000000c 3110           73 init:          srp     #WORK_REG
0000000e 8f             74                di                              ; disable int
0000000f e6f464         75                ld      t0,#%64                 ; 100 decimal
00000012 e6f5c9         76                ld      pre0,#11001001b         ; set t0 for 5 mS period
00000015 e6f27d         77                ld      t1,#125                 ;
00000018 e6f313         78                ld      pre1,#00010011b         ; set t1 for .5 mS period
0000001b e6f600         79                ld      p2m,#0                  ; outputs on p2
0000001e e6f701         80                ld      p3m,#1                  ; active on p2
00000021 e6f800         81                ld      p01m,#0                 ; outputs on p0
00000024 b000           82                clr     p0                      ; p0 low
00000026 b0ef           83                clr     sw_count                ; sw count = 0
00000028 e6f908         84                ld      ipr,#00001000b          ; make irq5 > irq3
0000002b e6fb30         85                ld      imr,#%30                ; enable irq4,irq5
0000002e e6ff80         86                ld      spl,#%80                ; set stack pointer
00000031 e6f10f         87                ld      tmr,#%0f                ; load and enable counters
00000034 2c04           88                ld      pointer,#%04            ; point to time regs
00000036 fc0c           89                ld      r15,#12                 ; six locations
00000038 b1e2           90 clear_reg:     clr     @pointer                ; clear ram
0000003a 2e             91                inc     pointer                 ;
0000003b fafb           92                djnz    r15,clear_reg           ; continue until all zero
0000003d d6Wwww         93                call    clear_buffer            ; clear buffer
00000040 e60812         94                ld      HOURS,#%12              ; start time at 12:00
00000043 d6Wwww         95                call    load_time               ; load starting time
00000046 2c20           96                ld      pointer,#BUFFER         ; start at top of buffer
00000048 e60003         97                ld      p0,#00000011b           ; take data and clear hi
0000004b 9f             98 main_loop:     ei                              ; enable interrupts
0000004c 8bfd           99                jr      main_loop               ;
                       100
                       101 ;*******************************************************************************;
                       102 ;            This interrupt routine updates the time                            ;
                       103 ;*******************************************************************************;
0000004e 70fd          104 time:          push    rp                      ; save current reg pointer
00000050 3100          105                srp     #TIME_REG               ; point to time reg group
00000052 d6Wwww        106                call    test_sw                 ; look at time-set switches
00000055 5e            107                inc     millisec                ; increment millisec reg
00000056 a6e5c8        108                cp      millisec,#200           ; one second?
00000059 7dWwww        109                jp      ult,exit_time           ; exit if not
0000005c b0e5          110                clr     millisec                ; set to zero
0000005e 760408        111                tm      STATUS,#00001000b       ; sw 1 pressed?
00000061 6b**          112                jr      z,test_sw2              ; no
00000063 060701        113                add     MINUTES,#1              ;
00000066 4007          114                da      MINUTES                 ; convert to bcd
00000068 a60760        115                cp      MINUTES,#%60            ; sixty minutes?
0000006b 7b**          116                jr      ult,inc_seconds         ;
0000006d b007          117                clr     MINUTES                 ;
```

APPENDIX (Continued)

```
0000006f 8b**           118                 jr      inc_seconds             ;
00000071 760410         119 test_sw2:       tm      STATUS,#00010000b       ; sw 2 pressed?
00000074 6b**           120                 jr      z,inc_seconds           ; no
00000076 060801         121                 add     HOURS,#1                ;
00000079 4008           122                 da      HOURS                   ;
0000007b a60813         123                 cp      HOURS,#%13              ; 1:00?
0000007e 7b**           124                 jr      ult,inc_seconds         ;
00000080 e60801         125                 ld      HOURS,#1                ;
00000083 06e601         126 inc_seconds:    add     seconds,#1              ; increment sec
00000086 40e6           127                 da      seconds                 ; convert to bcd
00000088 a6e660         128                 cp      seconds,#%60            ; sixty seconds?
0000008b 7b**           129                 jr      ult,exit_time           ; no
0000008d 460401         130                 or      STATUS,#00000001b       ; set message flag
00000090 b0e6           131                 clr     seconds                 ; set to zero
00000092 06e701         132                 add     minutes,#1              ; inc minutes
00000095 40e7           133                 da      minutes                 ; convert to bcd
00000097 a6e760         134                 cp      minutes,#%60            ; sixty minutes?
0000009a 7b**           135                 jr      ult,exit_time           ; not yet
0000009c b0e7           136                 clr     minutes                 ; set to zero
0000009e 06e801         137 set_hrs:        add     hours,#1                ; inc hours
000000a1 40e8           138                 da      hours                   ; convert to bcd
000000a3 a6e813         139                 cp      hours,#%13              ; 1:00?
000000a6 7b**           140                 jr      ult,exit_time           ; exit
000000a8 8c01           141                 ld      hours,#1                ; set to 1:00
000000aa d6Wvvv         142 exit_time:      call    time_convert            ; convert to individual chars
000000ad d6Wvvv         143                 call    load_time               ; load new values into buffer
000000b0 50fd           144                 pop     rp                      ; return to orig reg pointer
000000b2 bf             145                 iret                            ; return from int
                        146
                        147 ;*****************************************************************************;
                        148 ;               This routine converts the seconds, minutes, and hours bcd     ;
                        149 ;               data into units and tens-of-units for displaying              ;
                        150 ;*****************************************************************************;
000000b3 e40609         151 time_convert:   ld      SECONDS_LO,SECONDS      ; transfer contents
000000b6 e4060a         152                 ld      SECONDS_HI,SECONDS      ;
000000b9 56090f         153                 and     SECONDS_LO,#%0f         ; keep only lower bits
000000bc f00a           154                 swap    SECONDS_HI              ; swap nibbles
000000be 560a0f         155                 and     SECONDS_HI,#%0f         ; keep only lower bits
000000c1 e4070b         156                 ld      MINUTES_LO,MINUTES      ; transfer contents
000000c4 e4070c         157                 ld      MINUTES_HI,MINUTES      ;
000000c7 560b0f         158                 and     MINUTES_LO,#%0f         ; keep only lower bits
000000ca f00c           159                 swap    MINUTES_HI              ; swap nibbles
000000cc 560c0f         160                 and     MINUTES_HI,#%0f         ; keep only lower bits
000000cf e4080d         161                 ld      HOURS_LO,HOURS          ; transfer contents
000000d2 e4080e         162                 ld      HOURS_HI,HOURS          ;
000000d5 560d0f         163                 and     HOURS_LO,#%0f           ; keep only lower bits
000000d8 f00e           164                 swap    HOURS_HI                ;
000000da 560e0f         165                 and     HOURS_HI,#%0f           ; keep only lower bits
000000dd af             166                 ret                             ; return
                        167
                        168 ;*****************************************************************************;
                        169 ;           This subroutine loads the time data into the RAM buffer           ;
                        170 ;*****************************************************************************;
                        171
000000de 70fd           172 load_time:      push    rp                      ; save current reg pointer
000000e0 3110           173                 srp     #WORK_REG               ; point to working reg
000000e2 ac09           174                 ld      r10,#%09                ; load starting time reg
000000e4 bc20           175                 ld      r11,#BUFFER             ; load starting buffer reg
000000e6 e3ca           176 load_table:     ld      r12,@r10                ; load contents
000000e8 0c**           177                 ld      address_hi,#^hb number_table; load hi address of table
```

```
000000ea 1c**            178               ld      address_lo,#^lb number_table; load lo address of table
000000ec a6ec00          179               cp      r12,#0                ; is it zero?
000000ef 6b**            180               jr      eq,no_index           ; if yes, don't step thru table
000000f1 a0e0            181 index_num:    incw    address               ; step thru table
000000f3 a0e0            182               incw    address               ;
000000f5 a0e0            183               incw    address               ;
000000f7 a0e0            184               incw    address               ;
000000f9 a0e0            185               incw    address               ;
000000fb a0e0            186               incw    address               ;
000000fd caf2            187               djnz    r12,index_num         ; index if not zero
000000ff cc06            188 no_index:     ld      r12,#6                ; load no of segments
00000101 c3b0            189 load_time_reg: ldci   @r11,@address         ; load into reg
00000103 cafc            190               djnz    r12,load_time_reg     ; keep going if not zero
00000105 ae              191               inc     r10                   ; inc reg location
00000106 a6ea0f          192               cp      r10,#%0f              ; at ending reg?
00000109 7bdb            193               jr      ult,load_table        ; go again
0000010b e62b3a          194               ld      %2b,#%3a              ; put colon here
0000010e e6373a          195               ld      %37,#%3a              ; and here too
00000111 a60e00          196               cp      HOURS_HI,#0           ; zero?
00000114 eb**            197               jr      ne,load_time_ret      ;
00000116 ac00            198               ld      r10,#0                ; load zeros on last 5 columns
00000118 bc3e            199               ld      r11,#%3e              ; starting here
0000011a cc05            200               ld      r12,#5                ; blank out leading zero
0000011c f3ba            201 leading_loop: ld      @r11,r10              ;
0000011e be              202               inc     r11                   ; blank last five columns (hrs)
0000011f cafb            203               djnz    r12,leading_loop      ; step thru ram
00000121 50fd            204 load_time_ret: pop    rp                    ; return to time regs
00000123 af              205               ret                           ; return to caller
                         206
                         207
                         208 ;*************************************************************************;
                         209 ;      This subroutine checks to see if the time-set switches are pressed.    ;
                         210 ;*************************************************************************;
00000124 a803            211 test_sw:      ld      r10,p3                ; load sw data
00000126 60ea            212               com     r10                   ; 1's complement
00000128 56ea03          213               and     r10,#03               ; mask off upper bits
0000012b 760301          214               tm      p3,#1                 ; min pressed?
0000012e 6b**            215               jr      z,test_hrs            ; no
00000130 fe              216               inc     sw_count              ; inc counter
00000131 a6ef02          217               cp      sw_count,#2           ; debouncd?
00000134 7b**            218               jr      ult,exit_sw           ; not yet
00000136 460408          219               or      STATUS,#00001000b     ; set bit
00000139 8b**            220               jr      exit_sw               ; exit
0000013b 760304          221 test_hrs:     tm      p3,#4                 ; hrs pressed?
0000013e 6b**            222               jr      z,clear_sw            ; no
00000140 fe              223               inc     sw_count              ; inc debounce counter
00000141 a6ef02          224               cp      sw_count,#2           ; debounced?
00000144 7b**            225               jr      ult,exit_sw           ; not yet
00000146 460410          226               or      STATUS,#00010000b     ; set bit
00000149 af              227 exit_sw:      ret                           ; return to caller
0000014a 5604e7          228 clear_sw:     and     STATUS,#11100111b     ; reset sw status bits
0000014d b0ef            229               clr     sw_count              ; reset debounce counter
0000014f af              230               ret                           ; return
                         231
                         232 ;*************************************************************************;
                         233 ;      This is the timer interrupt routine.  When T1 hits TC, column data is  ;
                         234 ;     shifted one.                                                        ;
                         235 ;*************************************************************************;
00000150 70fd            236 shift:        push    rp                    ; save reg pointer
00000152 3110            237               srp     #WORK_REG             ; point to working reg
```

APPENDIX (Continued)

```
00000154 2e             238                 inc     pointer             ; point to next location in ram
00000155 8e             239                 inc     col_count           ;
00000156 a6e824         240                 cp      col_count,#36       ; thirty-six columns?
00000159 7b**           241                 jr      ult,test_flag       ; end of screen?
0000015b b0e8           242                 clr     col_count           ; reset col number
0000015d 2c20           243                 ld      pointer,#BUFFER     ; start at beginning of buffer
0000015f 460002         244                 or      p0,#00000010b       ; take data line high
00000162 760401         245 test_flag:      tm      STATUS,#00000001b   ; time to scroll message?
00000165 6b**           246                 jr      z,continue          ; no, display time
00000167 d6Wvvv         247                 call    load_message        ; load message data
0000016a 8b**           248                 jr      load_row            ; load row data
0000016c d6R000+00de,   249 continue:       call    load_time           ; get time data
0000016f e352           250 load_row:       ld      temp_1,@pointer     ; load contents
00000171 60e5           251                 com     temp_1              ;
00000173 5902           252                 ld      p2,temp_1           ; out to port
00000175 460004         253 clock_it:       or      p0,#00000100b       ; take clock hi
00000178 560001         254                 and     p0,#00000001b       ; take clock and data low
0000017b 50fd           255                 pop     rp                  ; restore reg pointer
0000017d bf             256                 iret                        ;
                        257 ;*****************************************************************************;
                        258 ;               This routine loads the message into the buffer                ;
                        259 ;*****************************************************************************;
0000017e 760402         260 load_message:   tm      STATUS,#00000010b   ; clear buffer?
00000181 eb**           261                 jr      nz,req_scroll       ;
00000183 d6Wvvv         262                 call    clear_buffer        ; clear contents of buffer
00000186 460402         263                 or      STATUS,#00000010b   ; set bit
00000189 0c**           264                 ld      address_hi,#^hb mess_beg      ;
0000018b 1c**           265                 ld      address_lo,#^lb mess_beg      ;
0000018d d6Wvvv         266                 call    get_ascii           ; get first character
00000190 9c06           267                 ld      seq_count,#6        ; start out with six segs
00000192 8b**           268                 jr      load_next_seg       ;
00000194 a6e900         269 req_scroll:     cp      seq_count,#0        ; six segments loaded?
00000197 eb**           270                 jr      ne,load_next_seg    ; no
00000199 760404         271                 tm      STATUS,#00000100b   ; end of message?
0000019c 6b**           272                 jr      z,load_next_char    ; load next ascii char
0000019e 6c00           273                 ld      temp_2,#0           ;
000001a0 d6Wvvv         274                 call    load_fifo           ; load data
000001a3 4e             275                 inc     zero_count          ; inc no of locations cleared
000001a4 a6e424         276                 cp      zero_count,#36      ; 36 locations yet?
000001a7 7b**           277                 jr      ult,scroll_return   ; load segment data
000001a9 b004           278                 clr     STATUS              ; display time now
000001ab b0e4           279                 clr     zero_count          ; clear counter
000001ad 8b**           280                 jr      scroll_return       ; return
000001af 9c06           281 load_next_char: ld      seq_count,#6        ; load 6 segments/character
000001b1 a0e0           282                 incw    address             ; point to next ascii char
000001b3 d6Wvvv         283                 call    get_ascii           ; get next character
000001b6 c26c           284 load_next_seg:  ldc     temp_2,@rr12        ; load seg from ascii table
000001b8 d6Wvvv         285                 call    load_fifo           ; shift the data
000001bb a0ec           286                 incw    rr12                ; point to next seg in table
000001bd 00e9           287                 dec     seq_count           ; decrement segment count
000001bf af             288 scroll_return:  ret                         ;
                        289
                        290 ;*****************************************************************************;
                        291 ;       This subroutine loads and shifts the RAM buffer                       ;
                        292 ;*****************************************************************************;
000001c0 fc24           293 load_fifo:      ld      r15,#36             ; load number of columns
000001c2 2c20           294                 ld      pointer,#BUFFER     ; load starting buffer reg
000001c4 e372           295 shift_fifo:     ld      temp_3,@pointer     ; get data
000001c6 f326           296                 ld      @pointer,temp_2     ; load new data
000001c8 68e7           297                 ld      temp_2,temp_3       ; transfer bytes
```

```
000001ca 2e                298               inc     pointer              ; next buffer address
000001cb faf7              299               djnz    r15,shift_fifo       ; load all columns
000001cd af                300               ret                          ; return to caller
                           301 ;*****************************************************************************;
                           302 ;   This subroutine indexes the character table and fetches the segment data. ;
                           303 ;*****************************************************************************;
000001ce cc**              304 get_ascii:     ld      r12,#^hb char_table  ; load starting add of table
000001d0 dc**              305               ld      r13,#^lb char_table  ;
000001d2 c2ea              306               ldc     r14,@rr10            ; load ascii data
000001d4 a6ee00            307               cp      r14,#0               ; end of message?
000001d7 eb**              308               jr      ne,load_next         ; no
000001d9 460404            309               or      STATUS,#00000100b    ; set bit to mark mess end
000001dc 8b**              310               jr      mess_return          ; return
000001de 26ee20            311 load_next:     sub     r14,#%20             ; subtract 20h
000001e1 a6ee00            312               cp      r14,#0               ; is it a space?
000001e4 6b**              313               jr      eq,mess_return       ; if yes, don't index table
000001e6 a0ec              314 index_table:   incw    rr12                 ; index table
000001e8 a0ec              315               incw    rr12                 ;
000001ea a0ec              316               incw    rr12                 ;
000001ec a0ec              317               incw    rr12                 ;
000001ee a0ec              318               incw    rr12                 ;
000001f0 a0ec              319               incw    rr12                 ;
000001f2 eaf2              320               djnz    r14,index_table      ; keep going if not zero
000001f4 af                321 mess_return:   ret                          ;
                           322
                           323 ;*****************************************************************************;
                           324 ;            This subroutine clears the RAM buffer.                         ;
                           325 ;*****************************************************************************;
000001f5 fc24              326 clear_buffer:  ld      r15,#36              ; get no of columns
000001f7 2c20              327               ld      pointer,#BUFFER      ; starting point of buffer
000001f9 b1e2              328 clear_loop:    clr     @pointer             ; clear contents
000001fb 2e                329               inc     pointer              ; next location
000001fc fafb              330               djnz    r15,clear_loop       ;  till out of columns
000001fe af                331               ret                          ; return to caller
                           332
                           333 ;*****************************************************************************;
                           334 ;          Message data area. Message can be up to 80 ASCII characters      ;
                           335 ;*****************************************************************************;
00000200                   336               .org    200h
00000200                   337 mess_beg:
00000200 5448495320444953  338               .asciz  'THIS DISPLAY IS POWERED BY ZILOG!'
00000208 504c415920495320
00000210 504f574552454420
00000218 4259205a494c4f47
00000220 2100
                           339
                           340 ;*****************************************************************************;
                           341 ;        This is the ASCII character look-up table.                         ;
                           342 ;*****************************************************************************;
00000280                   343               .org    280h
00000280                   344 char_table:
00000280 000000000000      345               .byte   0,0,0,0,0,0          ; space
00000286 0000007d00        346               .byte   0,0,0,7DH,0          ; !
0000028b 000070007000      347               .byte   0,0,70H,0,70H,0      ; "
00000291 00147f147f14      348               .byte   0,14H,7FH,14H,7FH,14H ; #
00000297 00122a7f2a24      349               .byte   0,12H,2AH,7FH,2AH,24H ; $
0000029d 006264081323      350               .byte   0,62H,64H,08H,13H,23H ; %
000002a3 003649350205      351               .byte   0,36H,49H,35H,02H,05H ; &
000002a9 000000700000      352               .byte   0,00,00,70H,00,00     ; '
000002af 001c22410000      353               .byte   0,1CH,22H,41H,0,0     ; (
```

APPENDIX (Continued)

```
000002b5 00000041221c   354                .byte   0,0,0,41H,22H,1CH     ; )
000002bb 0022147f1422   355                .byte   0,22H,14H,7FH,14H,22H ; *
000002c1 0008083e0808   356                .byte   0,08H,08H,3EH,08H,08H ; +
000002c7 000001060000   357                .byte   0,0,1,6,0,0           ; ,
000002cd 000808080808   358                .byte   0,8,8,8,8,8           ; -
000002d3 000000010000   359                .byte   0,0,0,1,0,0           ; .
000002d9 000204081020   360                .byte   0,2,4,8,10H,20H       ; /
                        361 ;numbers
000002df 003e4549513e   362 number_table:  .byte   0,3EH,45H,49H,51H,3EH ; 0
000002e5 0000217f0100   363                .byte   0,0,21H,7FH,01,0      ; 1
000002eb 002345494931   364                .byte   0,23H,45H,49H,49H,31H ; 2
000002f1 004241495966   365                .byte   0,42H,41H,49H,59H,66H ; 3
000002f7 000c14247f04   366                .byte   0,0CH,14H,24H,7FH,04H ; 4
000002fd 00725151514e   367                .byte   0,72H,51H,51H,51H,4EH ; 5
00000303 001e29494946   368                .byte   0,1EH,29H,49H,49H,46H ; 6
00000309 004047485060   369                .byte   0,40H,47H,48H,50H,60H ; 7
0000030f 003649494936   370                .byte   0,36H,49H,49H,49H,36H ; 8
00000315 003149494a3c   371                .byte   0,31H,49H,49H,4AH,3CH ; 9
                        372 ;    MORE SPECIAL CHARACTERS
0000031b 000000140000   373                .byte   0,0,0,14H,0,0         ; :
00000321 000001160000   374                .byte   0,0,1,16H,0,0         ; ;
00000327 000814224100   375                .byte   0,8,14H,22H,41H,0     ; <
0000032d 001414141414   376                .byte   0,14H,14H,14H,14H,14H ; =
00000333 000041221408   377                .byte   0,0,41H,22H,14H,08H   ; >
00000339 0020404d5020   378                .byte   0,20H,40H,4DH,50H,20H ; ?
                        379 ;    AT SIGN AND UPPERCASE LETTERS                ;
0000033f 003e415d4d39   380                .byte   0,3EH,41H,5DH,4DH,39H ; @
00000345 001f2444241f   381                .byte   0,1FH,24H,44H,24H,1FH ; A
0000034b 007f49494936   382                .byte   0,7FH,49H,49H,49H,36H ; B
00000351 003e41414122   383                .byte   0,3EH,41H,41H,41H,22H ; C
00000357 007f4141413e   384                .byte   0,7FH,41H,41H,41H,3EH ; D
0000035d 007f49494941   385                .byte   0,7FH,49H,49H,49H,41H ; E
00000363 007f48484840   386                .byte   0,7FH,48H,48H,48H,40H ; F
00000369 003e41414547   387                .byte   0,3EH,41H,41H,45H,47H ; G
0000036f 007f0808087f   388                .byte   0,7FH,08H,08H,08H,7FH ; H
00000375 0000417f4100   389                .byte   0,00H,41H,7FH,41H,00H ; I
0000037b 0002010101fe   390                .byte   0,02H,01H,01H,01H,7EH ; J
00000381 007f08142241   391                .byte   0,7FH,08H,14H,22H,41H ; K
00000387 007f01010101   392                .byte   0,7FH,01H,01H,01H,01H ; L
0000038d 007f2018207f   393                .byte   0,7FH,20H,18H,20H,7FH ; M
00000393 007f1008047f   394                .byte   0,7FH,10H,08H,04H,7FH ; N
00000399 003e4141413e   395                .byte   0,3EH,41H,41H,41H,3EH ; O
0000039f 007f48484830   396                .byte   0,7FH,48H,48H,48H,30H ; P
000003a5 003e4145423d   397                .byte   0,3EH,41H,45H,42H,3DH ; Q
000003ab 007f484c4a31   398                .byte   0,7FH,48H,4CH,4AH,31H ; R
000003b1 003249494926   399                .byte   0,32H,49H,49H,49H,26H ; S
000003b7 0040407f4040   400                .byte   0,40H,40H,7FH,40H,40H ; T
000003bd 007e0101017e   401                .byte   0,7EH,01H,01H,01H,7EH ; U
000003c3 007c0201027c   402                .byte   0,7CH,02H,01H,02H,7CH ; V
000003c9 007f020c027f   403                .byte   0,7FH,02H,0CH,02H,7FH ; W
000003cf 006314081463   404                .byte   0,63H,14H,08H,14H,63H ; X
000003d5 0060100f1060   405                .byte   0,60H,10H,0FH,10H,60H ; Y
000003db 004345495161   406                .byte   0,43H,45H,49H,51H,61H ; Z
000003e1 007f7f414141   407                .byte   0,7FH,7FH,41H,41H,41H ; [
000003e7 002010080402   408                .byte   0,20H,10H,08H,04H,02H ; \
000003ed 00414141f7f7   409                .byte   0,41H,41H,41H,7FH,7FH ; ]
000003f3 000408100804   410                .byte   0,04H,08H,10H,08H,04H ; ^
000003f9 000101010101   411                .byte   0,01H,01H,01H,01H,01H ; -
                        412
                        413                .end
```

DTMF TONE GENERATION USING THE Z8 CCP

IN MANY APPLICATIONS A MICROPROCESSOR MUST ACCESS PHONE LINES FOR COMMUNICATIONS OR DATA EXCHANGE. THE Z8'S PULSE WIDTH MODULATION CAPABILITIES ALLOW DTMF TONE GENERATION IN SOFTWARE.

INTRODUCTION & THEORY OF OPERATION

The program outlined below generates DTMF tones using a pulse width modulation (PWM) algorithm. PWM is used to vary the DC level of the output by varying the duty cycle (the "on" time divided by the cycle period) of the square wave. Varying the "on" time by a sine function and then feeding the output through a lowpass filter yields a sine wave.

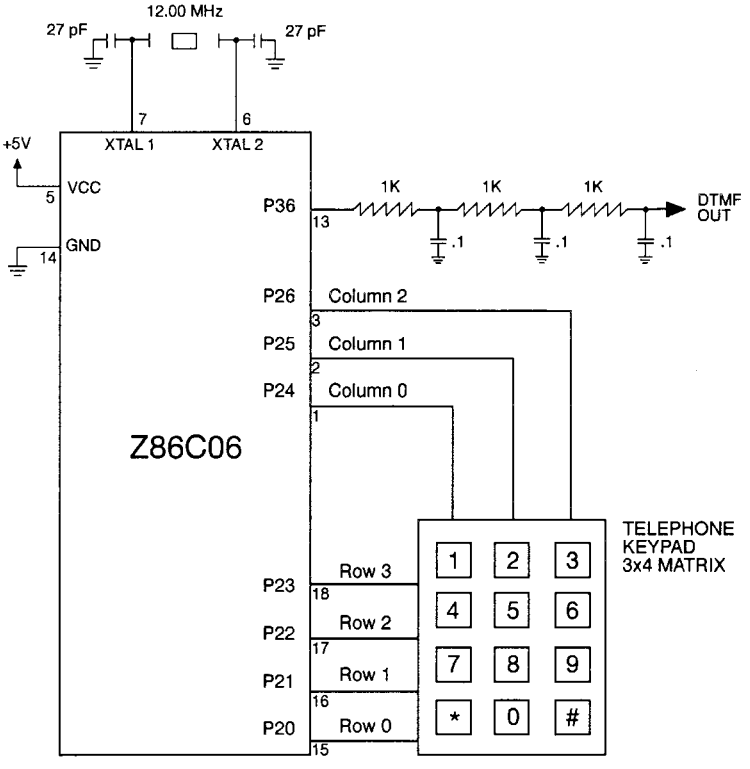
Sine values are determined by the Basic program in Listing 1. The sine table contains 256 entries, which in turn contain hexadecimal values representing a sine function for 360 degrees. These values are indexed and loaded into T1 at a sampling rate that, according to Nyquist, must be at least twice the highest frequency tone that we want to reproduce. Since the highest frequency for this application is 1477 Hz, the sampling rate must be at least twice this, or 2954 samples per second. The higher the sampling rate, the greater the accuracy. In the example illustrated here a sample rate of 12000 samples per second is used both for higher accuracy and ease in filtering.

Since we are, in effect, producing two tones, two pointers are used to fetch the next value in the look-up table: one for row frequencies, and one for column frequencies. The frequency of the resulting sine wave can be calculated by multiplying the number of steps in the sine look-up table (256) by the desired frequency, divided by the sampling rate. This offset value is added to the current pointer, which then fetches the next hex number from the look-up table. This is done for both the row and column frequencies. The two are added, then loaded into the timer register (T1).

The lowpass filter was chosen to have a corner frequency of the lowest column frequency, or 1209 Hz. At this point, the column frequencies will be at least 3 dB below the row frequencies. However, telephone lines themselves act as a large-scale lowpass filter, and by the time the tones reach the telephone switching equipment, the amplitude should be the same. The spec therefore calls for the column frequencies to be 3 dB higher than the row frequencies. In the telephone industry this is known as "twist."

This adjustment can be made in software by taking the hex value from the look-up table for the column frequency and doing a "rotate left." This results in twice the amplitude for the column frequency — 6 dB gain, or 3 dB up from the row frequency, just where we want to be. Overall dB level from the output of the lowpass filter can be adjusted with a potentiometer.

DTMF TONE GENERATION



LISTING 1

```
;---------------------------------------------------------------------------
;  This program, using the Zilog Z8, is designed to produce DTMF tones without
; an external DAC. The tones are produced using PWM, using one of the timers
; to vary the duty cycle of the output pulses. A sine look-up table is indexed
; at a sample rate well above the highest tone (1447 Hz). Both column and row
; values from the look-up table are added in software to produce the PWM output
; at the sample frequency. These hex values are loaded into a timer, which det-
; ermines the pulse width. At Terminal Count (TC), the port pin is taken low,
; and the difference between this period and the sample period is loaded into
; the timer. When run through a low-pass filter, the result is an accurate DTMF
; tone. Crystal frequency is 12.0000 MHz. The keypad columns are output from
; P24-6, while the row inputs are connected to P20-3.
;  This program was written on 6-2-92 by Don Owen Newquist, Zilog, Inc.
;---------------------------------------------------------------------------
WORK_REG1       .equ    10h
offset_hi       .equ    r0
offset_lo       .equ    r1
offset          .equ    rr0
row_inc_hi      .equ    r4
row_inc_lo      .equ    r5
row_inc         .equ    rr4
pointer_hi      .equ    r6
pointer_lo      .equ    r7
pointer         .equ    rr6
col_inc_hi      .equ    r8
col_inc_lo      .equ    r9
col_inc         .equ    rr8
r_freq_hi       .equ    r10
r_freq_lo       .equ    r11
r_freq          .equ    rr10
c_freq_hi       .equ    r12
c_freq_lo       .equ    r13
c_freq          .equ    rr12
row_val         .equ    r14
col_val         .equ    r15
xtal            .equ    12000
sample          .equ    12000
ctval           .equ    xtal/8/sample
tabstp          .equ    256

WORK_REG0       .equ    00h
bounce          .equ    r4
counter_1       .equ    r5
key_cnt         .equ    r6
key_temp        .equ    r7
temp_1          .equ    r8
scan            .equ    r9
```

```
BOUNCE          .EQU    %04
COUNTER_1       .EQU    %05
KEY_CNT         .EQU    %06
KEY_TEMP        .EQU    %07
TEMP_1          .EQU    %08
SCAN            .EQU    %09

                .org    0000h
                .word   0
                .word   0
                .word   0
                .word   0
                .word   key_scan
                .word   timer_1
;-------------------------------------------------------------------------------;
;                              INITIALIZATION                                   ;
;-------------------------------------------------------------------------------;
                .org    000ch
                di                      ; disable int
                ld      rp,#0f0h        ; point to
                ld      r0,#0feh        ; pcon reg
                ld      r11,#0          ; crystal div by 2
                srp     #WORK_REG1      ; lowest bank
                ld      p3m,#0          ; default
                ld      p01m,#04h       ; int stack
                ld      spl,#80h        ; stack at highest ram
                ld      irq,#00h        ;
                clr     imr             ; clear int mask
                ld      pre0,#05h       ; cont mode
                ld      t0,#00          ; sampling time = 125 microsec
                ld      pre1,#6h        ; one shot mode
                clr     p3              ; clear port 3
                ld      imr,#10h        ; enable irq4
                ld      ipr,#0bh        ; irq5 > irq4
                ld      tmr,#03h        ; load & en t0
                ld      pointer_hi,#^hb sine    ; get lookup table
                ld      pointer_lo,#^lb sine    ; address
                ld      r_freq_hi,#1fh          ; 697
                ld      r_freq_lo,#0efh
                ld      c_freq_hi,#17h  ; 1209
                ld      c_freq_lo,#26h
                ld      row_inc_hi,#00  ; 697
                ld      row_inc_lo,#01
                ld      col_inc_hi,#00  ; 1209
                ld      col_inc_lo,#01  ;
                ld      row_val,#10h    ;
                ld      col_val,#10h    ;
                clr     BOUNCE          ; debounce counter
                clr     KEY_CNT         ;
                clr     KEY_TEMP        ;
                clr     TEMP_1          ;
                clr     SCAN            ;
                ei                      ; enable interrupts
check_bounce:   jr      check_bounce    ;
```

```
;-------------------------------------------------------------------------;
;                      KEYBOARD SCAN ROUTINE                              ;
;-------------------------------------------------------------------------;
key_scan:       push    rp              ;
                srp     #WORK_REG0      ;
                ld      p2m,#00001111b  ; out for p27-p24, in p23-p20
                ld      scan,#11101111b ; load scan byte
load_scan:      ld      p2,scan         ; output to port 2
                nop                     ;
                nop                     ;
                ld      temp_1,p2       ; load contents of p2
                ld      counter_1,#0    ; load counter
row_loop:       inc     key_cnt         ;
                scf                     ;
                rrc     temp_1          ; rotate right
                jr      c,no_keys       ;
                cp      key_temp,key_cnt; same key?
                jr      ne,load_keys    ; not the same
                inc     bounce          ; increment bounce counter
                cp      bounce,#2       ; bounce = 2 ?
                jr      ult,load_keys   ;
                call    dtmf_out        ; call transmit routine
                jr      exit            ;
load_keys:      ld      key_temp,key_cnt; transfer key data
                clr     key_cnt         ;
                pop     rp              ;
                iret                    ;
no_keys:        inc     counter_1       ; looking for a one
                cp      counter_1,#4    ; keep looking if not all rows
                jr      ult,row_loop    ; look again at next row
                scf                     ; set carry flag
                rlc     scan            ; rotate to scan next column
                jr      c,load_scan     ; scan next column if no carry
exit:           clr     bounce          ;
                clr     key_cnt         ;
                clr     key_temp        ;
                clr     counter_1       ;
                pop     rp              ;
no_irq:         iret                    ;
```

```
;-------------------------------------------------------------------------------
;               Get Keyswitch Offset Values
;-------------------------------------------------------------------------------
dtmf_out:       push    rp
                srp     #WORK_REG1
                ld      tmr,#00h        ; reset t0 bits
                ld      imr,#20h        ; reset t0 vector
                ld      pre0,#4         ; one-shot mode
                ld      offset_hi,#^hb offset_tbl
                ld      offset_lo,#^lb offset_tbl
                sub     KEY_CNT,#1
                cp      KEY_CNT,#0
                jr      eq,no_index
index_loop:     incw    offset
                incw    offset
                incw    offset
                incw    offset
                dec     KEY_CNT
                cp      KEY_CNT,#0
                jr      ne,index_loop
no_index:       ldc     r_freq_hi,@offset
                incw    offset          ;
                ldc     r_freq_lo,@offset
                incw    offset          ;
                ldc     c_freq_hi,@offset
                incw    offset
                ldc     c_freq_lo,@offset
                ld      t0,#ctval
enable:         ei
;-------------------------------------------------------------------------------
;               DTMF Generation Routine
;-------------------------------------------------------------------------------
start:          or      p3,#40h         ;
                or      tmr,#0fh        ;
                tcm     p2,#0fh         ;
                jr      z,exit_dtmf     ;
dtmf_loop:      tm      irq,#10h        ;
                jr      z,dtmf_loop     ;
                and     irq,#0efh       ;
                jr      start           ;
exit_dtmf:      ld      imr,#10h        ; timer 0 int
                ld      pre0,#01h       ; cont mode, full count
                ld      imr,#10h        ; t0 only
                ld      tmr,#03h        ;
                ld      t0,#00          ;
                pop     rp              ;
                ret                     ;
```

```
;-------------------------------------------------------------------------------
;               Timer 1 Interrupt Routine
;-------------------------------------------------------------------------------
timer_1:        xor     p3,#40h         ; toggle port pin
                rcf
                add     row_inc_lo,r_freq_lo
                adc     row_inc_hi,r_freq_hi
                ld      pointer_lo,row_inc_hi
                ldc     row_val,@pointer
                add     col_inc_lo,c_freq_lo
                adc     col_inc_hi,c_freq_hi
                ld      pointer_lo,col_inc_hi
                ldc     col_val,@pointer
                rl      col_val
                add     row_val,col_val
load_t1:        ld      t1,row_val
                iret

                .org    180h
offset_tbl:
                .byte   1fh,0bfh,37h,70h
                .byte   23h,00h,37h,70h
                .byte   26h,0cfh,37h,70h
                .byte   2ah,0d7h,37h,70h
                .byte   1fh,0bfh,3dh,30h
                .byte   23h,00h,3dh,30h
                .byte   26h,0cfh,3dh,30h
                .byte   2ah,0d7h,3dh,30h
                .byte   1fh,0bfh,43h,0afh
                .byte   23h,00h,43h,0afh
                .byte   26h,0cfh,43h,0afh
                .byte   2ah,0d7h,43h,0afh

                .org    200h
sine:
        .byte   18h,18h,18h,18h,18h,19h,19h,19h,19h,19h,1ah,1ah
        .byte   1ah,1ah,1ah,1ah,1bh,1bh,1bh,1bh,1bh,1bh,1ch,1ch
        .byte   1ch,1ch,1ch,1ch,1dh,1dh,1dh,1dh,1dh,1dh,1dh,1dh
        .byte   1eh,1eh,1eh,1eh,1eh,1eh,1eh,1eh,1eh,1eh,1fh,1fh
        .byte   1fh,1fh,1fh,1fh,1fh,1fh,1fh,1fh,1fh,1fh,1fh,1fh
        .byte   1fh,1fh,1fh,1fh,1fh,1fh,1fh,1fh,1fh,1fh,1fh,1fh
        .byte   1fh,1fh,1fh,1fh,1fh,1fh,1fh,1fh,1fh,1fh,1fh,1eh
        .byte   1eh,1eh,1eh,1eh,1eh,1eh,1eh,1eh,1eh,1dh,1dh,1dh
        .byte   1dh,1dh,1dh,1dh,1ch,1ch,1ch,1ch,1ch,1ch,1ch,1bh
        .byte   1bh,1bh,1bh,1bh,1bh,1ah,1ah,1ah,1ah,1ah,1ah,19h
        .byte   19h,19h,19h,19h,18h,18h,18h,18h,18h,18h,17h,17h
        .byte   17h,17h,17h,16h,16h,16h,16h,16h,16h,15h,15h,15h
        .byte   15h,15h,15h,14h,14h,14h,14h,14h,14h,13h,13h,13h
        .byte   13h,13h,13h,13h,12h,12h,12h,12h,12h,12h,12h,12h
        .byte   12h,11h,11h,11h,11h,11h,11h,11h,11h,11h,11h,11h
        .byte   11h,11h,11h,11h,11h,11h,11h,11h,11h,11h,11h,11h
        .byte   11h,11h,11h,11h,11h,11h,11h,11h,11h,11h,11h,11h
        .byte   11h,11h,11h,11h,11h,11h,11h,11h,11h,11h,11h,11h
        .byte   12h,12h,12h,12h,12h,12h,12h,12h,12h,13h,13h,13h
        .byte   13h,13h,13h,13h,14h,14h,14h,14h,14h,14h,15h,15h
        .byte   15h,15h,15h,15h,16h,16h,16h,16h,16h,16h,17h,17h
        .byte   17h,17h,17h,18h

                .end
```

TIMEKEEPING WITH THE Z8

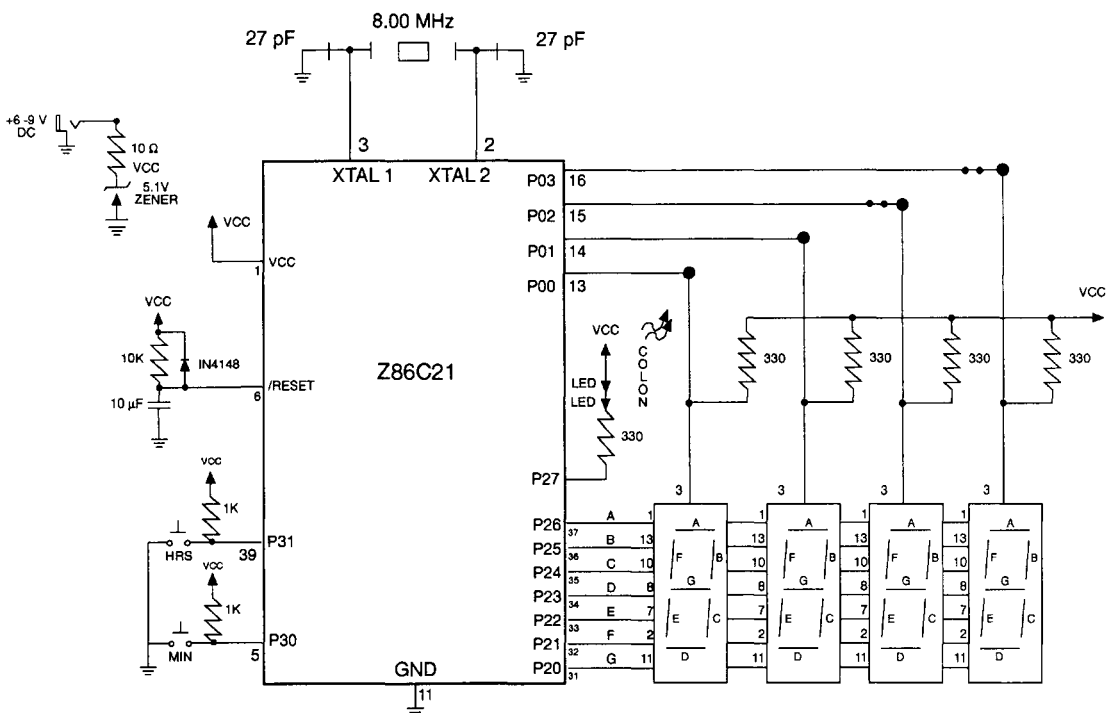
KEEP TRACK OF THE CURRENT TIME WITH THE Z8, USING MINIMAL HARDWARE. THIS APPLICATION NOTE DESCRIBES SEVERAL SOFTWARE-BASED METHODS OF TIMEKEEPING, WITH COMPLETE CODE LISTINGS.

INTRODUCTION

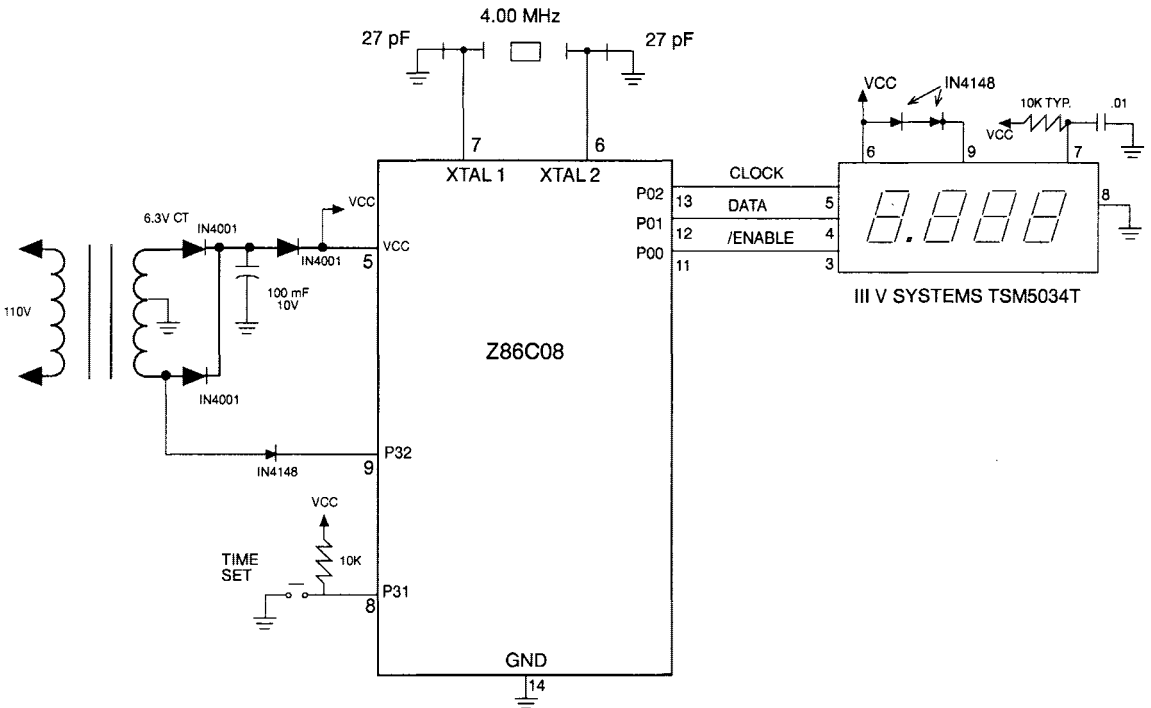
Many controller applications require ongoing tracking of the current time — at a specific time, for instance, the controller may have to take a specific action, like turning on a solenoid. This is accomplished by using a Real Time Clock, or RTC, the sole purpose of which is to keep track of time. A microprocessor-based system may use a RTC integrated circuit for timekeeping, or accomplish the same results through software, using the microprocessor's internal clock as a timebase. There are trade-offs associated with each approach, but the software method is by far the most economical, provided the micro can handle the added software overhead.

This Application Note covers two methods of timekeeping in software, with display of the current time on seven-segment LEDS. Time set switches are provided to advance the time. One method uses the microprocessor crystal as a timebase, while the other uses the highly-accurate 60 Hz line frequency as the timebase.

TIMEKEEPING BASED ON THE 8 MHZ CRYSTAL



TIMEKEEPING BASED ON THE 60 HZ LINE FREQUENCY



TIMEKEEPING BASED ON THE 8 MHZ CRYSTAL

```
;-------------------------------------------------------------------------
;       This program configures the Zilog Z8 to keep time using the 12.00 MHz
; crystal frequency as a timebase. The time is displayed on four (4) seven-
; segment, common-anode LEDs. The cathodes of the displays are driven by P2.
; The anodes of the displays are connected to P0, and a pull-up resistor. The
; colon, made up of two discrete LEDs, flashes at a 1 Hz rate, and is driven by
; P2-7. The set switches are connected to P3. This program may be adapted to
; any 28-pin or 40-pin Z8. Written by Don Owen Newquist, Zilog, on 5-1-91.
;-------------------------------------------------------------------------
WORK_REG        .equ    10h
address_hi      .equ    r0
address_lo      .equ    r1
address         .equ    rr0
pointer         .equ    r2
data            .equ    r3
col_count       .equ    r4
ring_counter    .equ    r5
HALF_SECOND     .equ    16h

TIME_REG        .equ    00h
millisec        .equ    r5
seconds         .equ    r6
minutes         .equ    r7
hours           .equ    r8
seconds_lo      .equ    r9
seconds_hi      .equ    r10
minutes_lo      .equ    r11
minutes_hi      .equ    r12
hours_lo        .equ    r13
hours_hi        .equ    r14
sw_count        .equ    r15
STATUS          .equ    %04
MILLISEC        .equ    %05
SECONDS         .equ    %06
MINUTES         .equ    %07
HOURS           .equ    %08
SECONDS_LO      .equ    %09
SECONDS_HI      .equ    %0a
MINUTES_LO      .equ    %0b
MINUTES_HI      .equ    %0c
HOURS_LO        .equ    %0d
HOURS_HI        .equ    %0e

                .org    00
                .word   00
                .word   00
                .word   00
                .word   00
                .word   time
                .word   load_time
                .org    000ch
```

```
;*****************************************************************************;
;                  Initialization                                             ;
;*****************************************************************************;
init:           srp     #WORK_REG
                di                              ; disable int
                ld      t0,#250                 ; 250 decimal
                ld      pre0,#01010001b         ; set t0 for 5 mS period
                ld      pre1,#%13               ;
                ld      t1,#00                  ; set for .250 mS refresh
                ld      p2m,#0                  ; outputs on p2
                ld      p3m,#0                  ; open drain on p2
                ld      p01m,#04                ; outputs on p0, int stack
                clr     p0                      ; p0 low
                clr     p3                      ; p3 low
                clr     sw_count                ; sw count = 0
                clr     STATUS                  ; clear status reg
                clr     HALF_SECOND             ;
                ld      ipr,#00001000b          ; make irq5 > irq3
                ld      imr,#%30                ; enable irq4,irq5
                ld      spl,#%80                ; set stack pointer
                clr     sph                     ;
                ld      tmr,#%0f                ; load and enable counters
                ld      pointer,#04             ; point to time regs
                ld      r15,#12                 ; six locations
clear_reg:      clr     @pointer                ; clear ram
                inc     pointer                 ;
                djnz    r15,clear_reg           ; continue until all zero
                ld      HOURS,#%12              ; start time at 12:00
                ld      pointer,#HOURS_HI       ; start at hours reg
                ld      ring_counter,#%88       ; enable first digit
main_loop:      ei                              ; enable interrupts
                jr      main_loop               ;
;*****************************************************************************;
;               This routine converts the seconds, minutes, and hours bcd     ;
;               data into units and tens-of-units for displaying              ;
;*****************************************************************************;
time_convert:   ld      SECONDS_LO,SECONDS      ; transfer contents
                ld      SECONDS_HI,SECONDS      ;
                and     SECONDS_LO,#%0f         ; keep only lower bits
                swap    SECONDS_HI              ; swap nibbles
                and     SECONDS_HI,#%0f         ; keep only lower bits
                ld      MINUTES_LO,MINUTES      ; transfer contents
                ld      MINUTES_HI,MINUTES      ;
                and     MINUTES_LO,#%0f         ; keep only lower bits
                swap    MINUTES_HI              ; swap nibbles
                and     MINUTES_HI,#%0f         ; keep only lower bits
                ld      HOURS_LO,HOURS          ; transfer contents
                ld      HOURS_HI,HOURS          ;
                and     HOURS_LO,#%0f           ; keep only lower bits
                swap    HOURS_HI                ;
                and     HOURS_HI,#%0f           ; keep only lower bits
                ret                             ; return
```

```
;*****************************************************************************;
;             This interrupt routine updates the time                         ;
;*****************************************************************************;
time:           push    rp                      ; save current reg pointer
                srp     #TIME_REG               ; point to time reg group
                call    test_sw                 ; look at time-set switches
                inc     millisec                ; increment millisec reg
                cp      millisec,#100           ; half second?
                jr      ult,exit_time           ; don't toggle colon
                tm      STATUS,#00000001b       ; sw 1 pressed?
                jr      z,test_sw2              ; no
                add     MINUTES,#1              ;
                da      MINUTES                 ; convert to bcd
                cp      MINUTES,#%60            ; sixty minutes?
                jr      ult,inc_half_sec        ;
                clr     MINUTES                 ;
                jr      inc_half_sec            ;
test_sw2:       tm      STATUS,#00000010b       ; sw 2 pressed?
                jr      z,inc_half_sec          ; no
                add     HOURS,#1                ;
                da      HOURS                   ;
                cp      HOURS,#%13              ; 1:00?
                jr      ult,inc_half_sec        ;
                ld      HOURS,#1                ;
inc_half_sec:   inc     HALF_SECOND             ;
                clr     millisec                ;
                xor     p2,#%80                 ; toggle colon bit now
one_second:     cp      HALF_SECOND,#2          ; one second?
                jp      ult,exit_time           ; exit if not
                clr     HALF_SECOND             ; set to zero
inc_seconds:    add     seconds,#1              ; increment sec
                da      seconds                 ; convert to bcd
                cp      seconds,#%60            ; sixty seconds?
                jr      ult,exit_time           ; no
                clr     seconds                 ; set to zero
                add     minutes,#1              ; inc minutes
                da      minutes                 ; convert to bcd
                cp      minutes,#%60            ; sixty minutes?
                jr      ult,exit_time           ; not yet
                clr     minutes                 ; set to zero
set_hrs:        add     hours,#1                ; inc hours
                da      hours                   ; convert to bcd
                cp      hours,#%13              ; 1:00?
                jr      ult,exit_time           ; exit
                ld      hours,#1                ; set to 1:00
exit_time:      call    time_convert            ; convert to individual bcd
                pop     rp                      ; return to orig reg pointer
                iret                            ; return from int
```

```
;*****************************************************************************;
;      This subroutine checks to see if the time-set switches are pressed.    ;
;*****************************************************************************;
test_sw:        push    rp                      ;
                srp     #WORK_REG               ;
                ld      data,p3                 ; get switch data
                com     data                    ; invert data
                and     data,#%03               ; only first two bits
                cp      data,#0                 ;
                jr      eq,clear_sw             ;
                tm      data,#1                 ; min pressed?
                jr      z,test_hrs              ; no
                inc     sw_count                ; inc counter
                cp      sw_count,#2             ; debouncd?
                jr      ult,exit_sw             ; not yet
                or      STATUS,#00000001b       ; set bit
                jr      exit_sw                 ; exit
test_hrs:       tm      data,#2                 ; hrs pressed?
                jr      z,clear_sw              ; no
                inc     sw_count                ; inc debounce counter
                cp      sw_count,#2             ; debounced?
                jr      ult,exit_sw             ; not yet
                or      STATUS,#00000010b       ; set bit
exit_sw:        pop     rp                      ;
                ret                             ; return to caller
clear_sw:       clr     STATUS                  ; reset sw status bits
                clr     sw_count                ; reset debounce counter
                pop     rp                      ;
                ret                             ; return
;*****************************************************************************;
;            This subroutine loads the time data into the RAM buffer          ;
;*****************************************************************************;
load_time:      push    rp                      ; save current reg pointer
                srp     #WORK_REG               ; point to working reg
load_table:     ld      r12,@pointer            ; load contents
                ld      address_hi,#^hb led_table; load hi address of table
                ld      address_lo,#^lb led_table; load lo address of table
                cp      r12,#0                  ; is it zero?
                jr      eq,no_index             ; if yes, don't step thru table
index_num:      incw    address                 ; step thru table
                djnz    r12,index_num           ; index if not zero
no_index:       lde     data,@address           ; load segments
                and     p2,#%80                 ;
                or      p2,data                 ; load port 2 with segments
                ld      p3,ring_counter         ; turn on digit
                dec     pointer                 ; inc reg location
                cp      pointer,#SECONDS_HI     ; at ending reg?
                jr      ugt,load_time_ret       ; exit
                ld      pointer,#HOURS_HI       ; start at beginning
                cp      @pointer,#0             ;
                jr      ne,load_time_ret        ;
                dec     pointer                 ; don't display leading zero
                rr      ring_counter            ;
load_time_ret:  rr      ring_counter            ; rotate counter
                pop     rp                      ; return to time regs
                iret                            ; return from interrupt
```

```
led_table:
                .byte   00111111B       ; ZERO
                .byte   00000110B       ; ONE
                .byte   01011011B       ; TWO
                .byte   01001111B       ; THREE
                .byte   01100110B       ; FOUR
                .byte   01101101B       ; FIVE
                .byte   01111101B       ; SIX
                .byte   00000111B       ; SEVEN
                .byte   01111111B       ; EIGHT
                .byte   01100111B       ; NINE

                .end
```

TIMEKEEPING BASED ON THE 60 HZ LINE FREQUENCY

```
;-------------------------------------------------------------------------
;   This clock routine uses the Zilog Z86E08 to keep time. The 60 Hz line
; frequency causes an interrupt to the CPU every half cycle using one of the
; on-board comparators. The time registers are then incremented, and the dis-
; play refreshed at 1/ 60 second intervals. The time is displayed on a Three-
; Five Systems TSM6X34 Four Digit Display. The display has serial data and
; clock inputs, along with on-board display drivers. Port P0 provides the
; clock, data, and reset lines for the display. Port P2 is available for
; user options. The time set switch is connected to the second comparator
; input, and when pressed, advances the time at a 60 Hz rate.
;   This program was written by Don Owen Newquist on May 16, 1992.
;-------------------------------------------------------------------------
TIME_REG        .equ    10h
counter         .equ    r0
bit_count       .equ    r1
address_hi      .equ    r4
address_lo      .equ    r5
address         .equ    rr4
pointer         .equ    r6
data            .equ    r7
milliseconds    .equ    r8
seconds         .equ    r9
minutes         .equ    r10
hours           .equ    r11
minutes_lo      .equ    r12
minutes_hi      .equ    r13
hours_lo        .equ    r14
hours_hi        .equ    r15

BUFFER          .equ    04h
START           .EQU    02
ENABLE_HI       .EQU    01
ONE             .EQU    02
ZERO            .EQU    %FD
CLOCK_HI        .EQU    04
CLOCK_LO        .EQU    %FB

                .org    00
                .word   00
                .word   00
                .word   00
                .word   time
                .word   00
                .word   00
                .org    000ch
```

```
;*****************************************************************************
;                Initialization                                               *
;*****************************************************************************
init:           srp     #TIME_REG
                di                              ; disable int
                ld      p2m,#0                  ; outputs on p2
                ld      p3m,#0                  ; open drain on p2
                ld      p01m,#04                ; outputs on p0, int stack
                clr     p0                      ; p0 low
                clr     p3                      ; p3 low
                clr     irq                     ;
                ld      imr,#%08                ; enable irq3
                ld      spl,#%80                ; set stack pointer
                clr     sph                     ;
                ld      pointer,#18h            ; point to time regs
                ld      counter,#6              ; six locations
clear_reg:      clr     @pointer                ; clear ram
                inc     pointer                 ;
                djnz    counter,clear_reg       ; continue until all zero
                ld      hours,#%12              ; start time at 12:00
                ld      pointer,#minutes_lo     ; start at minutes reg
                call    time_convert            ;
                call    load_time               ;
main_loop:      ei                              ; enable interrupts
                jr      main_loop               ;
;*****************************************************************************
;             This interrupt routine updates the time                         *
;*****************************************************************************
time:           tm      p3,#2                   ; sw pressed?
                jr      nz,inc_minutes          ; no
                inc     milliseconds            ;
                cp      milliseconds,#60        ; one second yet?
                jr      ult,exit_time           ;
                clr     milliseconds            ;
                inc     seconds                 ;
                cp      seconds,#60             ;
                jr      ult,exit_time           ;
                clr     seconds                 ;
inc_minutes:    add     minutes,#1              ; inc minutes
                da      minutes                 ; convert to bcd
                cp      minutes,#%60            ; sixty minutes?
                jr      ult,exit_time           ; not yet
                clr     minutes                 ; set to zero
set_hrs:        add     hours,#1                ; inc hours
                da      hours                   ; convert to bcd
                cp      hours,#%13              ; 1:00?
                jr      ult,exit_time           ; exit
                ld      hours,#1                ; set to 1:00
exit_time:      call    time_convert            ; convert to individual bcd
                call    load_time               ;
                iret                            ; return from int
```

```
;*****************************************************************************;
;               This routine converts the seconds, minutes, and hours bcd     ;
;               data into units and tens-of-units for displaying              ;
;*****************************************************************************;
time_convert:   ld      minutes_lo,minutes      ; transfer contents
                ld      minutes_hi,minutes      ;
                and     minutes_lo,#%0f         ; keep only lower bits
                swap    minutes_hi              ; swap nibbles
                and     minutes_hi,#%0f         ; keep only lower bits
                ld      hours_lo,hours          ; transfer contents
                ld      hours_hi,hours          ;
                and     hours_lo,#%0f           ; keep only lower bits
                swap    hours_hi                ;
                and     hours_hi,#%0f           ; keep only lower bits
                ret                             ; return

;*****************************************************************************;
; This subroutine loads the time data into the RAM buffer,then to the display ;
;*****************************************************************************;

load_time:      ld      pointer,#minutes_lo     ; load contents
                ld      BUFFER,#04              ; load buffer add
                ld      counter,#4              ;
                ld      address_hi,#^hb led_table; load hi address of table
load_table:     ld      address_lo,#^lb led_table; load lo address of table
                ld      data,@pointer           ;
                add     address_lo,data         ;
no_index:       ldc     data,@address           ; load segments
                ld      @BUFFER,data            ;
                inc     pointer                 ;
                inc     BUFFER                  ;
                djnz    counter,load_table      ;
                ld      BUFFER,#4               ;
                ld      counter,#3              ;
                ld      p0,#START               ; clock & enable low, data hi
send_start:     call    clock_out               ; clock the data
                djnz    counter,send_start      ;
                ld      counter,#4              ;
next_digit:     ld      bit_count,#8            ;
                ld      data,@BUFFER            ;
rotate:         rcf                             ;
                rrc     data                    ;
                jr      nc,zero                 ;
                or      p0,#ONE                 ;
                jr      clock_it                ;
zero:           and     p0,#ZERO                ;
clock_it:       call    clock_out               ;
                djnz    bit_count,rotate        ;
                inc     BUFFER                  ;
                djnz    counter,next_digit      ;
                call    clock_out               ; one more to load data
                ld      p0,#ENABLE_HI           ; take enable line high
                ret
```

```
;-------------------------------------------------------------------------
;               Clock Out LED Data
;-------------------------------------------------------------------------
clock_out:      or      p0,#CLOCK_HI            ; set hi
                nop                             ; delay
                nop                             ; delay
                nop                             ; delay
                and     p0,#CLOCK_LO            ; take clock lo
                nop                             ;
                nop                             ;
                nop                             ;
                ret                             ; return from interrupt

                .org    %00f0
led_table:
                .byte   00111111B               ; ZERO
                .byte   00000110B               ; ONE
                .byte   01011011B               ; TWO
                .byte   01001111B               ; THREE
                .byte   01100110B               ; FOUR
                .byte   01101101B               ; FIVE
                .byte   01111101B               ; SIX
                .byte   00000111B               ; SEVEN
                .byte   01111111B               ; EIGHT
                .byte   01100111B               ; NINE

                .end
```

SINE TABLE

```
05 ' Listing 1
10 '-----------------------------------------------------------------
20 '            *** SIN-TAB.BAS ***
30 '
40 '  Purpose= to generate a complete sine table
50 '  of hex numbers in format... [TAB].byte[TAB]01h,02h,...,0Ch
60 '-----------------------------------------------------------------
70 '
80 ON ERROR GOTO 590
90 DIM B$(256)
100 DIM C$(22)
110 DEFINT A,B
120 PI = 3.141593
130 C=360/256
140 GOTO 390
150 FOR I=0 TO 255
160 H=B+A*SIN((C*I*PI)/180)
170 A$=HEX$(H)                    'HEX$ function rounds off H autom'ly
180 IF LEN(A$)>2 THEN PRINT "--Bad data: some bytes bigger than FFh":GOTO 430
190 IF LEN(A$)=1 THEN A$="0"+A$          'ADD LEADING ZERO, IF NECESS.
200 B$(I+1)=A$+"h"
210 NEXT I
220 I=1
230 FOR J=1 TO 22
240 C$(J)=CHR$(9)+".byte"+CHR$(9)        'START W/[TAB].byte[TAB]
250 FOR K=1 TO 12
260 C$(J)=C$(J)+B$(I)+","
270 I=I+1:IF I>256 GOTO 290
280 NEXT K
290 L=LEN(C$(J))-1
300 C$(J)=LEFT$(C$(J),L)                 'KILL LAST COMMA
310 NEXT J
320 OPEN "TABLE.ASC" FOR OUTPUT AS #1
330 FOR J=1 TO 22:PRINT #1,C$(J):NEXT J
340 CLOSE #1
350 BEEP:PRINT:PRINT "SINE DATA STORED IN CURRENT DRIVE & DIRECTORY"
360 PRINT "IN ASCII FILE CALLED    TABLE.ASC  ":PRINT
370 PRINT "   ( Press any key )"
380 T$=INKEY$:IF T$="" THEN 380 ELSE SYSTEM
390 CLS:LOCATE 10,35:PRINT "* SIN-TAB *":PRINT
400 PRINT TAB(10);"Generates a 256-byte sine table in the current"
410 PRINT TAB(10);"directory by creating an ASCII file called: TABLE.ASC"
420 PRINT TAB(10);"(If the file already exists, it will be overwritten !!!)"
430 PRINT TAB(10);"Sine wave is of the form:  A sin(xt) where A= amplitude."
440 PRINT TAB(10);"What is amplitude ? (range= 1 - 127)  A= ";
450 LINE INPUT T$
460 A=VAL(T$):IF A<1 OR A>127 THEN GOTO 430
470 PRINT TAB(10);"An offset is needed to keep all values positive."
480 PRINT TAB(15);"Suggest:";A+1
490 PRINT TAB(20);"Is this okay ? (Y/N)"
```

```
500 T$=INKEY$:IF T$="" GOTO 500
510 IF T$=CHR$(27) THEN SYSTEM
520 IF T$="Y" OR T$="y" THEN B=A+1:PRINT "WAIT...":GOTO 150
530 IF T$="N" OR T$="n" THEN 540 ELSE 500
540 PRINT TAB(10);"Enter new value (0-255): ";
550 LINE INPUT T$
560 B=VAL(T$):IF B<0 OR B>255 THEN PRINT "--Illegal value--":GOTO 470
570 PRINT "WAIT...":GOTO 150
580 '          *** ERROR TRAP ***
590 IF ERL<>320 GOTO 630
600 BEEP:PRINT:PRINT "Unable to create (or write to) file: TABLE.ASC"
610 PRINT "    ( Press any key )"
620 RESUME 380
630 IF ERL=460 THEN PRINT "--Illegal value--":PRINT:RESUME 440
640 IF ERL=560 THEN PRINT "--Illegal value--":PRINT:RESUME 470
650 BEEP:PRINT:PRINT "*** ERROR ";ERR;" OCCURRED AT LINE";ERL:PRINT
660 PRINT "    ( Press any key )"
670 RESUME 380
```

SERIAL COMMUNICATIONS USING THE Z8 CCP SOFTWARE UART

MANY APPLICATIONS REQUIRE ASYNCHRONOUS COMMUNICATIONS WITH THE OUTSIDE WORLD. THIS APPLICATION NOTE PRESENTS ONE METHOD FOR ACCOMPLISHING SERIAL COMMUNICATIONS THROUGH SOFTWARE RATHER THAN HARDWARE BY TAKING ADVANTAGE OF THE FLEXIBILITY OF THE Z8® MCU CCP™.

INTRODUCTION

Straightforward serial communications between Zilog Z8 CCP processors and the outside world are possible using the 9600 baud software UART described in this Application Note. This technique is particularly well suited to projects which require asynchronous communications but where a hardware UART is cost- or space-prohibitive. Designed to be used with the Zilog CCP family running at an 8 MHz clock frequency, this approach requires the use of only one counter/timer and two port pins.

THEORY OF OPERATION

Essentially, the UART remains in an idle loop until it either (a) senses an interrupt request on port P3-1 (Receive), or (b) it senses a character in the BUFFER register to be transmitted. 11 bits in total are counted: one start bit, eight data bits, one parity bit, and one stop bit. A typical schematic is provided in Figure 1.

Retro Workshop 果粉工作室

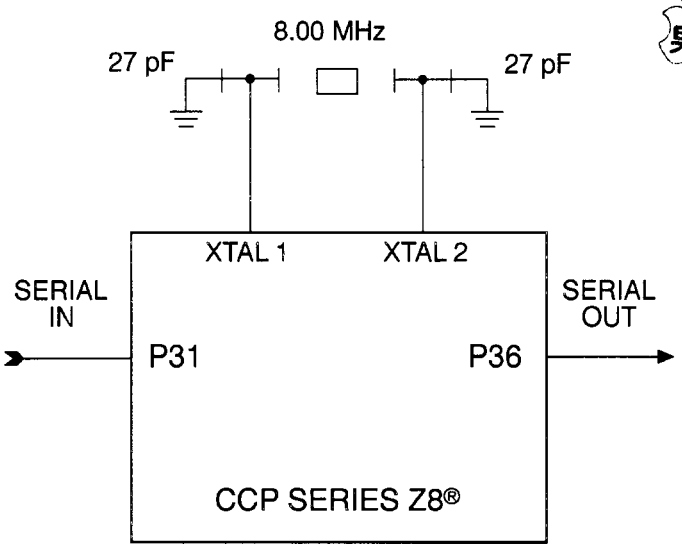


Figure 1. Serial CCP UART

RECEIVE MODE

The UART goes into operation upon sensing a receive character via an interrupt (low-going edge) request on P3-3. It then prepares to receive one start bit (R10), 8 data bits (R10), one parity bit, and one stop bit (R10). Subsequently, it programs the TO timer as a down-counter which reaches a terminal count in 0.104 mS. This is the sampling rate. The interrupt vector is loaded with the receive subroutine address, which is jumped to when an IRQ 4 interrupt is generated. This routine decrements the bit counters, and rotates the data in register R10. When a byte is assembled, it waits for the parity and stop bit, which then completes the transfer.

TRANSMIT MODE

The UART is ready to transmit at any time, and senses a transmit character when BUFFER is not zero. When BUFFER is loaded with data to be transmitted, the jump vector is loaded with the transmit subroutine address. The counter/timer TO is then loaded for a terminal count at 0.104 mS. This is the sampling rate for the transmission. When TO times out, an IRQ 4 interrupt is generated. The program jumps to the IRQ 4 interrupt routine, and then immediately jumps to the transmit routine. The transmit routine decrements the bit counter and rotates out the transmit data.

```
;*********************************************************************
;*                                                                   *
;*      SERIAL DATATRANSFER APPLICATION FOR CCP-SERIES               *
;*      WITHOUT HARDWARE UART.                                       *
;*                                                                   *
;*      WRITTEN BY W. MANSFELD, ZILOG GERMANY / feb. 1991/ dec.91    *
;*                                                                   *
;*********************************************************************
TO_VECTOR_H:    .EQU    %20
TO_VECTOR_L:    .EQU    %21
BYTE_BUFF:      .EQU    %22     ;CONTAINS Rx RETURN BYTE ( RP=20 )

                .ORG    %00

                .WORD   DUMMY
                .WORD   DUMMY
                .WORD   DUMMY
                .WORD   SER_IN          ;RX BYTE INTERRUPT
                .WORD   TO_INT          ;T0 JUMP VECTOR INTERRUPT
                .WORD   DUMMY

                .ORG    %0C

                DI
                LD      SPL,#%7E
                CLR     SPH
                LD      IPR,#1          ;INT. PRIORITY
                SRP     #%20            ;SET REG. POINTER
                                        ;refer to upper EQU's

                LD      P3M,#00000001B  ;PORT 3 NORMAL MODE
                LD      P01M,#00000100B ;SELECT INTERNAL STACK

;               PORTS 0,1,2 NOT USED IN THIS APP.

                LD      IMR,#00011000B  ;ENABLE T0,RX INT
                EI

;       SEND A ENDLESS BYTE STREAM VIA Tx LINE

TEST_LOOP:
                LD      R4,#'5'         ;ASCII 5  TEST BYTE
                CALL    SER_OUT         ;SEND BYTE
                JR      TEST_LOOP
```

```
;*********************************************************************
;       SERIAL RECEIVE ON INTERRUPT P3,0                              *
;       INTERRUPT WHEN A CHARACTER COMES FROM HOST                    *
;       input: none                                                   *
;*********************************************************************
SER_IN:

;       SAVE USED REGISTERS

        PUSH    R15             ; TRANSPARENT /DATA BUFFER
        PUSH    R6              ; SERIAL BIT COUNT ++

        PUSH    IMR             ;SAVE INT. MODE REGISTER
        LD      IMR,#00010000B  ;SELECTE T0 INT. ONLY


;       PUT A DELAY LOOP TO THAT POINT IF BAUDRATE IS BELOW OR
;       9600 BAUD, TO REACH MIDDLE OF START BIT BEFORE STARTING
;       RX BIT WINDOW



;       SERIAL BAUDRATE
;       USE T0 FOR BIT-CENTRE INTERRUPT

        LD      PRE0,#00000111B ;MODIFY PRESCALER PRE0
        LD      T0,#104         ;BAUD RATE = 9600 / ON 8MHz CRYSTAL
        LD      TMR,#00000011B  ;START/PRESET T0

;       MODIFY T0 INT. JUMP VECTOR

        LD      R7,#^HB SERIN_T0_INT
        LD      T0_VECTOR_H,R7
        LD      R7,#^LB SERIN_T0_INT
        LD      T0_VECTOR_L,R7
        EI
        LD      R6,#8           ;READ 8 DATABYTES
        CLR     R15             ;CLEAR RECEIVESHIFT REGISTER(BUFFER)
SER_IN1:
        CP      R6,#0           ;8 BITS RECEIVED ?
        JR      NE,SER_IN1      ;WAIT FOR BITS RECEIVED
        LD      BYTE_BUFF,R15   ;SAVE RX BYTE IN BUFFER

        LD      R6,#1           ;READ 1 STOPBIT
        CLR     R15
SER_IN2:
        CP      R6,#0           ;READ  DATA DONE?
        JR      NE,SER_IN2      ;WAIT FOR BIT RECEIVED
        DI
        AND     TMR,#11111100B  ;T0 OFF
        POP     IMR             ;INTERRUPT MASK BACK
        CLR     IRQ
        POP     R6
        POP     R15
        IRET                    ;RX DONE
```

```
;*********************************************************
;*      OUTPUT A BYTE THROUGH THE SERIAL                 *
;*      INPUT : R4 CONTAINS TX-DAT                       *
;*      OUTPUT: R4 HOLDS INPUT DATA                      *
;*********************************************************
SER_OUT:
;       SAVE USED REGISTERS
        PUSH    R5              ;T0 INT. VECTOR POINTER LB
        PUSH    R7              ;TX DATA
        PUSH    R10             ;BIT COUNTER
        PUSH    IMR             ;FOR T0 int. ONLY ACTIVE HEREIN
        PUSH    R4              ;SAVE DATA
        LD      IMR,#00010000B  ;T0 int. ON

;       SERIAL BAUDRATE, USE T0 FOR TIMING

        LD      T0,#104         ; BAUD RATE = 9600
        LD      PRE0,#00000101B ; PRESCALER VALUE 1
        LD      TMR,#%03        ; LOAD + ENABLE T0

;       MODIFY T0 INT. JUMP VECTOR FOR Tx

        LD      R7,#^HB SER_OUT_T0_INT
        LD      T0_VECTOR_H,R7
        LD      R7,#^LB SER_OUT_T0_INT
        LD      T0_VECTOR_L,R7
        EI

;       NOW SEND DUMMY 0-bit and one START BIT

        LD      R10,#2          ;SEND 2 BIT: 0 + START-BIT
        LD      R7,#11111101B   ;VALUE for the 2 bits
SER_OUT1:
        CP      R10,#0          ;DONE?
        JR      NZ,SER_OUT1
        LD      R10,#8          ;SEND 8 BITS OF DATA
        POP     R7              ;TAKE DATABYTE FROM STACK
        PUSH    R7              ;CORRECT THE STACK ( PUT BACK THAT BYTE )
;       CHECK IF ALL DATABITS SENT
SER_OUT2:
        CP      R10,#0          ;DONE?
        JR      NZ,SER_OUT2

;       NOW SEND STOP-BIT

        LD      R10,#1          ;SEND  ONE BIT ( STOP BIT )
        LD      R7,#%FF         ;VALUE
SER_OUT3:
        CP      R10,#0          ;DONE?
        JR      NZ,SER_OUT3
        DI
        AND     TMR,#11111100B  ;STOP T0
        POP     R4
        POP     IMR             ;OLD IMR BACK
        POP     R10
        POP     R7
        POP     R5
        EI
        RET                     ;RET ONLY, WAS NOT AN INT. SERVICE
```

```
;************************************************
;*      TRANSMIT T0 INTERRUPT HANDLER           *
;*      FOR  BIT GENERATION TO P3,7             *
;************************************************
SER_OUT_T0_INT:
                DEC     R10             ;BIT CNT DOWN
                RR      R7              ;SHIFT BIT
                JR      C,SER_O_T0_INT1 ;BIT = 1
                AND     P3,#%7F         ;BIT = 0
                IRET
SER_O_T0_INT1:  OR      P3,#%80         ;BIT = 1
                IRET

;       PROGRAMMABLE USE OF T0 ( RX or TX )

T0_INT:
                JP      @T0_VECTOR_H    ;INDIRECT T0 int JUMP
                                        ;FOR Tx,RX bit read
DUMMY:          IRET

;************************************************
;*      RECEIVE  T0 INTERRUPT HANDLER           *
;*      FOR  BIT READ FROM P3,0                 *
;************************************************
SERIN_T0_INT:
                DEC     R6      ;BIT COUNT
                PUSH    R15     ;SAVE FOR TEMPORARY USE
                LD      R15,P3  ;READ THE PORT / BIT  Rx
                RR      R15     ;CHECK if 0 or 1
                POP     R15     ;NOW USE AS RECEIVE SHIFT BUFFER
                RRC     R15     ;SHIFT THE CARRY =0 OR 1 INTO BUFFER
                IRET


                .END
```

INTERFACING LCDS TO THE Z8

BY TRADING HARDWARE APPROACHES FOR SOFTWARE SOLUTIONS, INTERFACING A Z8® MCU TO A M1641 LCD MODULE BECOMES A PRACTICAL AND SIMPLIFIED DESIGN METHODOLOGY.

INTRODUCTION

There has been an increasing demand for interfacing Liquid Crystal Displays (LCDs) to low-end microcontrollers in recent years. Unfortunately, little has been offered to address real-world applications and to help the design engineer understand how to make LCDs work. This App Note (Application Note) explains and shows a software method of interfacing a Z8 to an LCD module. The challenge to the programmer is the fixed amount of ROM space. Although almost any Z8 device is usable, the CCP™ (Consumer Controller Processor) Family is referenced; For example, the Z86C40, Z86C96/61, etc.

OVERVIEW

Since the Z8 architecture is so flexible, it would be very difficult to include all possible applications in the spec sheet example Figures and Tables. Since the purpose of this App Note is to reduce the complexity of interfacing Z8s with LCDs, the software routines are intended to be "cut and paste," so choose the ones that meet your needs (caution, do not forget the initialization routines). Also, remember that not all possible functions inside the LCD were utilized. In many applications, it is only necessary to transfer ASCII for display in 16-character chunks. For messages containing more than 16 characters, the message may be broken down into two 16-character fields, each alternately being displayed. The details on special functions are in the manufacturers data sheets.

M1641 LCD MODULE

The M1641 LCD Module is a 1-line by 16-character display with an on-board controller; the HD44780. The HD44780 divides the 16 characters into two lines of eight characters each. Even though this is a two-line device, physically, all characters appear on the same line. The controller has an on-board character generator in ROM capable of displaying 192 ASCII characters, along with eight user programmable characters. All characters are displayed in a 5x7 font.

INTERFACE

The LCD module can be connected to either an 8-bit or 4-bit data bus. There are three control lines: RS, R//W, and E (Enable). The RS line selects either an instruction (Low), or data (High). The R//W line (write active Low), allows data to be written to the LCD (Low) or read from the LCD (High).

The Enable line (E) is used to latch data to and from the LCD (Figure 1a and 1b and Table 1a and 1b). The V1c line is used for adjusting the contrast, but for most applications may be tied to ground.

Table 1a. Read Characteristics

Item		Symbol	Standard		Unit
			Min.	Max.	
Enable cycle time		$t_{CYC}E$	1000	-	ns
Enable pulse width	High level	PW_{EH}	450	-	ns
Enable rise and fall time		t_{Er}, t_{Ef}	-	25	ns
Setup time	RS, R//W—E	t_{AS}	140	-	ns
Address hold time		t_{AH}	10	-	ns
Data delay time		t_{DDR}	-	320	ns
Data hold time		t_{H}	20	-	ns

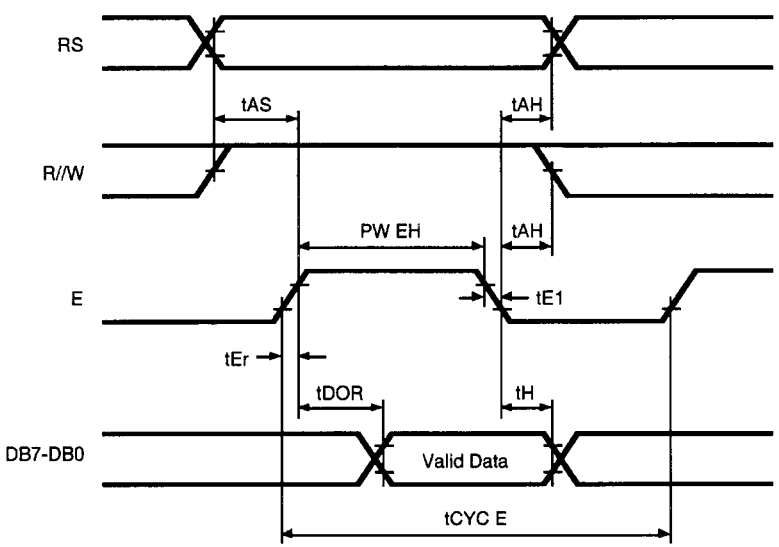


Figure 1a. LCD Read* Timing

Note:
*V_{DD}=5.0V ±5%, V_{SS}=0V, T_A=0°C to 50°C

Table 1b. Write Characteristics

Item		Symbol	Standard		Unit
			Min.	Max.	
Enable cycle time		$t_{CYC}E$	1000	-	ns
Enable pulse width	High level	PW_{EH}	450	-	ns
Enable rise and fall time		t_{Er}, t_{Ef}	-	25	ns
Setup time	RS, R/W—E	t_{AS}	140	-	ns
Address hold time		t_{AH}	10	-	ns
Data delay time		t_{DDR}	195	-	ns
Data hold time		t_{H}	10	-	ns

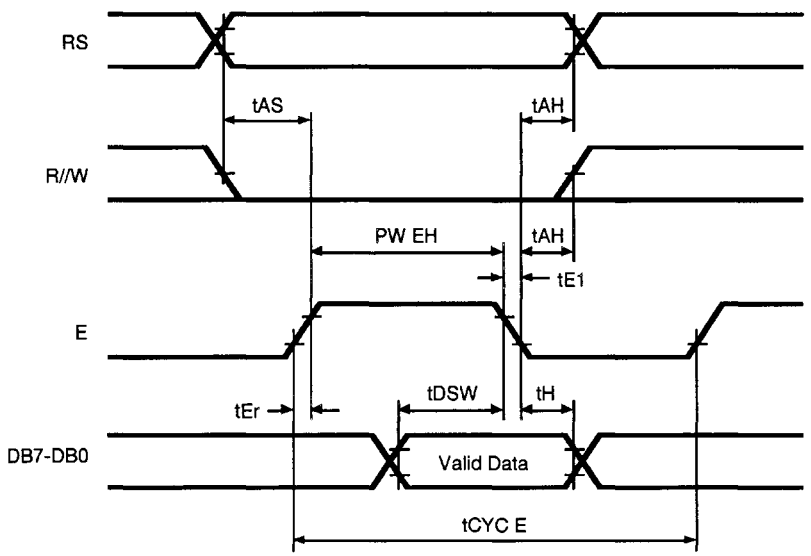


Figure 1b. LCD Write* Timing

Note:
*V_{DD}=5.0V ±5%, V_{SS}=0V, T_A=0°C to 50°C

INTERFACE (Continued)

V_{SS} is tied to ground while V_{CC} is tied to the +5 Volt supply. Unless your application is heavily "I/O bound," it is easiest to use one of the Z8's 8-bit ports for data, and use two lines from Port 3 for control. Figure 2 shows a typical Z8 interface.

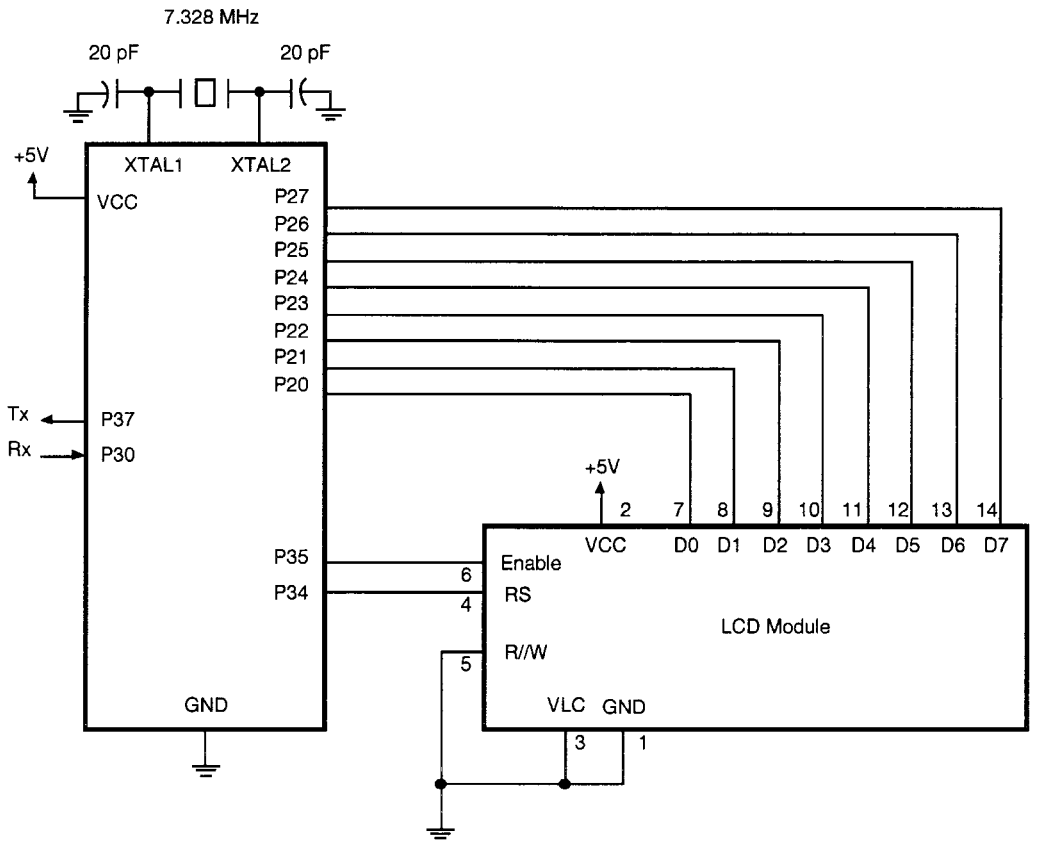


Figure 2. Typical Z8/LCD Interface

INITIALIZATION

After power up, initialize the LCD before sending data. Note that the LCD module is very slow. Therefore, it is necessary to write a delay loop program in between instructions. Again, the LCD can be configured for either 8-bit or 4-bit data transfer. When operating in 4-bit mode, the upper nibble gets transferred first, followed by the lower nibble.

Table 2 shows complete instruction codes. In order to write the instruction codes to the LCD module, the RS line must be Low. Figure 3 shows an initialization sequence for an 8-bit transfer operation. The starting address for the DD RAM is 80H for the first eight characters. For the next eight characters, the starting address is COH. If the DD RAM is programmed for auto increment, then the DD RAM address is automatically incremented after each character write.

Table 2. LCD Instructions Codes

Instruction	Set		Instruction Code								Description	Execution Time (when fCP or fOSC is 250 kHz)
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear Display	0	0	0	0	0	0	0	0	0	1	Clears all display memory and returns the cursor to the home position (Address 0).	82 µs ~ 1.64 ms
Return Home	0	0	0	0	0	0	0	0	1	*	Returns the cursor to the home position (Address 0). Also returns the display being shifted to the original position. DD RAM contents remain unchanged.	40 µs ~ 1.6 ms
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S	Sets the cursor move direction and specifies or not to shift the display. These operations are performed during data write and read.	40 µs ~ 1.64 ms
Display ON/OFF Control	0	0	0	0	0	0	1	D	C	B	(D) is display ON/OFF control, memory remains unchanged in OFF condition. (C) cursor ON/OFF (B) blinking cursor.	40 µs
Cursor or Display Shift	0	0	0	0	0	1	S/C	R/L	*	*	Moves the cursor and shifts the display without changing DD RAM contents.	40 µs
Function Set	0	0	0	0	1	DL	N	F	*	*	Sets interface data length (DL), number of display lines (N), and character font (F).	40 µs
Set CG RAM Address	0	0	0	1	ACG						Sets the CG RAM address. CG RAM data is sent and received after this setting.	40 µs
Set DD RAM Address	0	0	1	ADD							Sets the DD RAM address. DD RAM data is sent and received after this setting.	40 µs
Read Busy Flag & Address	0	1	BF	AC							Reads Busy Flag (BF) indicating internal operation is being performed and reads address counter contents.	1 µs
Write Data to CG or DD RAM	1	0	Write Data								Writes into DD RAM or CG RAM.	40 µs
Read Data from CG or DD RAM	1	1	Read Data								Reads data from DD RAM or CG RAM.	40 µs

INITIALIZATION (Continued)

Notes to the previous table:

1. *Doesn't Matter

2. DD RAM :	Display data RAM	I/D=1:	Increment	C=1:	Cursor On	R/L=1:	Right shift
CG RAM:	Character generator RAM	I/D=0:	Decrement	C=0:	Cursor Off	R/L=0:	Left shift
A_{CG}:	CG RAM address	S=1:	Display shift	B=1:	Blink ON	DL=1:	8 bits
A_{DD}:	DD RAM address Corresponds to cursor address	S=0:	No display shift	B=0:	Blink OFF	DL=0:	4 bits
		D=1:	Display ON	S/C=1:	Display shift	N=1:	2 lines (M1641)
AC:	Address counter used for both of DD and CG RAM address	D=0:	Display OFF	S/C=0:	Cursor movement	N=0:	1 line (M24111 & L4041)
				BF=1:	Internal operation in progress	F=1:	5x10 dot-matrix (M24111 & L4041)
				BF=0:	Instruction can be accepted	F=0:	5x7 dot-matrix

3. Execution times in Table 2 indicate the maximum values when operating frequency is 250 kHz.
4. When f_{OSC} is 270 kHz: 40 µs x 250/270 = 37 µs

Display Initialization

Each time the module is turned on or reset, an initialization procedure must be executed. The procedure consists of sending a sequence of hex codes from the microprocessor or parallel I/O port. The initialization sequence turns on the cursor, clears the display, and sets the module onto an auto-increment mode.

The inital hex code 30, 34, or 38 is sent two or more times to ensure the module enters the 8-bit or 4-bit data mode. All the initialization sequences are performed under the condition of Register Select (RS) = 0 (Low) and Read/Write (R/W) = 0 (Low).

4-bit data bus microcontroller may operate the display module by sending the initialization sequence in 4-bit format. Since 4-bit operation requires the data to be sent twice over the higher 4-bit bus lines (D4-D7), memory requirements are doubled.

Example for the module with 5x7 character format under 8-bit data transfer.

Applicable modules: M1641, L1651, M1632, L1642, L1652, L2012, L2432, L4042, L1614, L2014, M4024

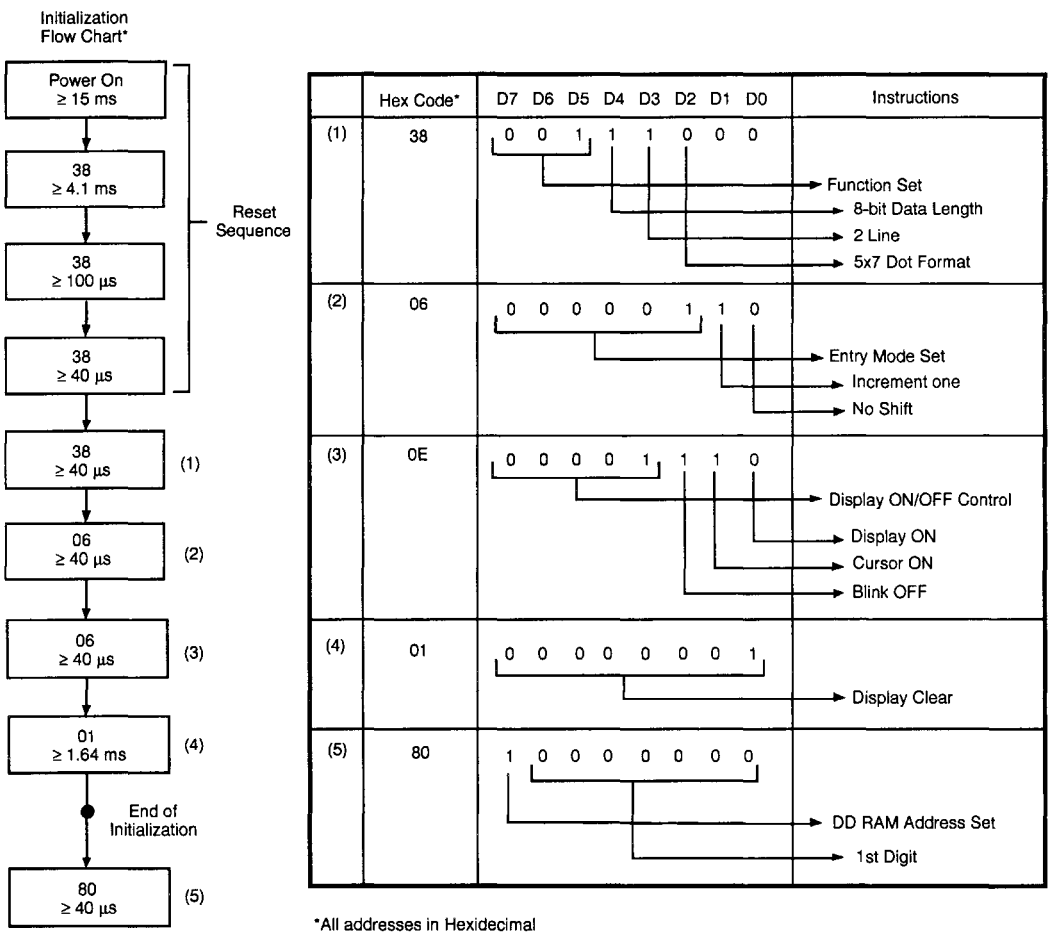

Initialization Flow Chart*
Power On ≥ 15 ms
38 ≥ 4.1 ms
38 ≥ 100 µs
38 ≥ 40 µs
Reset Sequence
38 ≥ 40 µs (1)
06 ≥ 40 µs (2)
06 ≥ 40 µs (3)
01 ≥ 1.64 ms (4)
End of Initialization
80 ≥ 40 µs (5)
Hex Code*
D7 D6 D5 D4 D3 D2 D1 D0
Instructions
(1) 38 0 0 1 1 1 0 0 0
Function Set
8-bit Data Length
2 Line
5x7 Dot Format
(2) 06 0 0 0 0 0 1 1 0
Entry Mode Set
Increment one
No Shift
(3) 0E 0 0 0 0 1 1 1 0
Display ON/OFF Control
Display ON
Cursor ON
Blink OFF
(4) 01 0 0 0 0 0 0 0 1
Display Clear
(5) 80 1 0 0 0 0 0 0 0
DD RAM Address Set
1st Digit
*All addresses in Hexidecimal

Figure 3. Initialization Sequence

Notes:
1. Both RS and R/W terminals shall be "0" in this sequence.
2. RS, R/W and Data are latched at the falling edge of the Enable signal, (falling edge is typically 10 ns; MAX: 25 ns).
3. M4024 has to be initialized on E1 and E2, respectively. Refer to (2) in Sample Programs.

LCD/Z8 PROGRAM

Appendix A (Listing 1) has all the routines for communicating to an LCD module. For this App Note exercise, there is no cursor, auto-increment for the DD RAM address, and no display shift. Since there is no read instruction from the LCD, the R/W is tied Low.

Transferring ASCII to the Z8 via the UART requires buffering before writing to the LCD, since the LCD is not able to keep up with the Z8 at high baud rates. Therefore, the 16 ASCII are first stored within one of the working register groups, then transferred to the LCD. Actually, the Z8 has the ability to store several messages within the register file. This is useful for applications that require a handful of commonly used messages that are downloaded from a master MPU after initialization.

Of course, messages can be stored in the internal Z8 ROM space. Since the program occupies less than 750 bytes, there is plenty of room to display other custom messages.

This program allows the Z8 user to interface to a 1-line by 16-character Liquid Crystal Display (LCD). It includes a routine to load serial data from the UART to the register file, then out to the display. For displaying "canned" messages send a control character followed by the message number. Another control character is used for accepting ASCII from the outside world followed by sixteen ASCII characters.

This program is intended to be used with the industry-standard 1-line by sixteen-character LCD module (Seiko M1641, etc.). Certain modifications are neccessary for 2-line modules (see manufactuers data sheets).

APPENDIX A
LCD Program (Listing 1)

```
;          M1641 Pinout                                                    ;
;          ------------                                                    ;
;           1 - Vss                                                        ;
;           2 - Vdd                                                        ;
;           3 - Vlc                                                        ;
;           4 - RS                                                         ;
;           5 - R//W                                                       ;
;           6 - E                                                          ;
;           7 - DB0                                                        ;
;           8 - DB1                                                        ;
;           9 - DB2                                                        ;
;          10 - DB3                                                        ;
;          11 - DB4                                                        ;
;          12 - DB5                                                        ;
;          13 - DB6                                                        ;
;          14 - DB7                                                        ;
;-------------------------------------------------------------------------;
CUR_HOME  .EQU   02H
DIS_CLEAR .EQU   01H
CG_RAM    .EQU   40H
DD_RAM_1  .EQU   80H
DD_RAM_2  .EQU   0C0H
PORT_2    .EQU   02H
PORT_0    .EQU   00H
CONST_1   .EQU   1FH
CONST_2   .EQU   0FFH
POLL_SIO  .EQU   08H
IRQ3_RES  .EQU   0F7H
STACK     .EQU   80H
CLOCK_LO  .EQU   0FEH
CLOCK_HI  .EQU   01H
RS_LOW    .EQU   0FDH
RS_HIGH   .EQU   02H
E_LOW     .EQU   0F8H
E_HIGH    .EQU   04H
MYREG_0   .EQU   00H
8_BITS    .EQU   38H
AI_NS     .EQU   06H
DO_NC     .EQU   0CH
BUFF_BEG  .EQU   10H
BUFF_END  .EQU   20H
MASK      .EQU   01H

MYREG2    .EQU   20H
;
          .ORG     0000H
          .WORD    0,0,0
          .word    RXD
          .word    0
          .word    0
          .org     000ch
```

APPENDIX A (Continued)

```
;-------------------------------------------------------------------------;
;                       INITIALIZATION                                    ;
;-------------------------------------------------------------------------;
BEGIN:
          DI                              ; DISABLE INTERRUPTS
          SRP     #MYREG_0                ; POINT TO REGS 10 - 1F HEX
          LD      SPL,#STACK              ; INITIALIZE STACK POINTER
          LD      P01M,#00                ; CONFIGURE P0 FOR OUTPUT
          LD      P2M,#0                  ; CONFIGURE P2 FOR OUTPUT
          LD      P3M,#%41                ; ACTIVE PULL-UPS FOR P2
          LD      IPR,#%28                ; IRQ3 IS HIGHEST
          LD      IMR,#%88                ; ENABLE IRQ3
          LD      IRQ,#0                  ; CLEAR IRQ
          CALL    LCD_INIT                ; INITIALIZE THE LCD
          EI                              ; ENABLE INTERRUPTS
ENABLE:
          JR      ENABLE                  ; WAIT FOR INTERRUPTS
;
;-------------------------------------------------------------------------;
;          INITIALIZE THE LIQUID CRYSTAL DISPLAY                          ;
;-------------------------------------------------------------------------;
LCD_INIT:
          AND     PORT_0,#RS_LOW          ; MAKE SURE P3-4 IS 0
          LD      PORT_2,#00              ; MAKE P2 ZERO
          OR      PORT_0,#E_HIGH          ; TAKE ENABLE HIGH
          AND     PORT_0,#E_LOW           ; BRING ENABLE BACK LOW
          CALL    WAIT_1                  ; WAIT AWHILE
          LD      PORT_2,#8_BITS          ; SET DATA LENGTH FOR 8 BITS
          OR      PORT_0,#E_HIGH          ; TAKE ENABLE HIGH
          AND     PORT_0,#E_LOW           ; BRING ENABLE BACK LOW
          CALL    WAIT_1                  ; WAIT AWHILE
          OR      PORT_0,#E_HIGH          ; TAKE ENABLE HIGH
          AND     PORT_0,#E_LOW           ; BRING ENABLE BACK LOW
          CALL    WAIT_1                  ; WAIT AWHILE
          OR      PORT_0,#E_HIGH          ; TAKE ENABLE HIGH
          AND     PORT_0,#E_LOW           ; BRING ENABLE BACK LOW
          CALL    WAIT_1                  ; WAIT AWHILE
          LD      PORT_2,#DO_NC           ; TURN ON DISPLAY, NO CURSOR!
          OR      PORT_0,#E_HIGH          ; TAKE ENABLE HIGH
          AND     PORT_0,#E_LOW           ; BRING ENABLE BACK LOW
          CALL    WAIT_1                  ; WAIT AWHILE
          LD      PORT_2,#AI_NS           ; ENTRY MODE: INC ADDRESS,NO SHIFT
          OR      PORT_0,#E_HIGH          ; TAKE ENABLE HIGH
          AND     PORT_0,#E_LOW           ; BRING ENABLE BACK LOW
          CALL    WAIT_1                  ; WAIT AWHILE
```

```
LCD_RES:
          LD     PORT_2,#DIS_CLEAR ; CLEAR DISPLAY
          OR     PORT_0,#E_HIGH    ; TAKE ENABLE HIGH
          AND    PORT_0,#E_LOW     ; BRING ENABLE BACK LOW
          CALL   WAIT_1            ; WAIT AWHILE
          LD     PORT_2,#CUR_HOME  ; CURSOR HOME
          OR     PORT_0,#E_HIGH    ; TAKE ENABLE HIGH
          AND    PORT_0,#E_LOW     ; BRING ENABLE BACK LOW
          CALL   WAIT_1            ; WAIT AWHILE
          LD     PORT_2,#CG_RAM    ; SET CG RAM
          OR     PORT_0,#E_HIGH    ; TAKE ENABLE HIGH
          AND    PORT_0,#E_LOW     ; BRING ENABLE BACK LOW
          CALL   WAIT_2            ; WAIT AWHILE
          LD     PORT_2,#DD_RAM_1  ; SET DD RAM
          OR     PORT_0,#E_HIGH    ; TAKE E HIGH
          AND    PORT_0,#E_LOW     ; TAKE E LOW
          CALL   WAIT_2            ; WAIT AWHILE
          RET

;-------------------------------------------------------------------------;
;           RECEIVE INTERRUPT ROUTINE - RECEPTION OF 1B HEX IS FOLLOWED   ;
;           BY SIXTEEN ASCII CHARACTERS TO BE DISPLAYED ON LCD MODULE.     ;
;           RECEPTION OF 1C HEX, FOLLOWED BY A MESSAGE NUMBER (e.g. 0-9),;
;           DISPLAYS ONE OF THE INTERNAL MESSAGES.                        ;
;-------------------------------------------------------------------------;
RXD:
          DI                       ; DISABLE INTERRUPTS
          PUSH   RP                ; SAVE REG POINTER
          SRP    #MYREG2           ; POINT TO WORKING REGS
          LD     R7,SIO            ; GET BYTE FROM SIO
          CP     R7,#%1B           ; ESCAPE CHARACTER?
          JR     NE,FS             ; NO, TRY FS CHARACTER
          CALL   ASCII             ; GET READY FOR TEXT
          JR     RXDOUT            ; EXIT
FS:
          CP     R7,#%1C           ; IF FS, THEN CANNED MESSAGE
          JR     NE,RXDOUT         ; IF NOT, EXIT
          CALL   INT_MSG           ; CALL CANNED MESSAGE ROUTINE
RXDOUT:
          POP    RP                ; RESTORE POINTER
          IRET                     ; RETURN FROM INTERRUPT
;
;-------------------------------------------------------------------------;
;         ACCESS ONE OF THE INTERNAL LCD MESSAGES                         ;
;-------------------------------------------------------------------------;
INT_MSG:
          PUSH   RP                ; SAVE CURRENT REG POINTER
          SRP    #MYREG_0          ; POINT TO REG 0 - FH
          PUSH   IMR               ; SAVE CURRENT IMR
          DI                       ; DISABLE INTERRUPTS
          LD     IMR,#%20          ; ENABLE IRQ5 ONLY
          CLR    IRQ               ; CLEAR ANY PENDING
          EI                       ; ENABLE INTERRUPTS
```

APPENDIX A (Continued)

```
MSG_NUM:
          TM     IRQ,#POLL_SIO       ; POLL SIO FOR NEXT BYTE
          JR     Z,MSG_NUM           ; KEEP POLLING
          AND    IRQ,#IRQ3_RES       ; CLEAR IRQ3
          LD     R4,SIO              ; LOAD BYTE FROM SIO
          LD     R8,^HB LCD_MSG      ; LOAD LOOKUP TABLE ADD
          LD     R9,^LB LCD_MSG      ;
          LD     R7,#BUFF_BEG        ; GET BUFFER START ADD
LOOKUP_1:
          LD     R6,#16              ; SET BYTE COUNTER FOR 16
LOOKUP_2:
          INCW   RR8                 ; STEP TO DESIRED MESSAGE
          DJNZ   R6,LOOKUP_2         ; IF NOT 0, KEEP DECREMENTING
          DJNZ   R4,LOOKUP_1         ; INDEX MESSAGES
LOAD_MSG:
          LDEI   @R7,@RR8            ; LOAD INT BUFFER, INCREMENT
          CP     R7,#BUFF_END        ; END OF BUFFER?
          JR     LT,LOAD_MSG         ; NO, KEEP LOADING
          CALL   LCD_RES             ; SETUP FOR LCD TRANSFER
          CALL   LCD_LOAD            ; GO TO LOAD ROUTINE
          POP    IMR                 ; RESTORE INTERRUPT STRUCTURE
          POP    RP                  ; RESTORE REG POINTER
          RET                        ; RETURN TO CALLER

;-------------------------------------------------------------------------;
;          LOAD ASCII CHARACTERS FOR DISPLAY ON LCD MODULE                 ;
;-------------------------------------------------------------------------;
ASCII:
          PUSH   IMR                 ; SAVE CONTENTS OF IMR
          DI                         ; DISABLE INTERRUPTS
          LD     IMR,#%20            ; ENABLE IRQ 5 ONLY
          CLR    IRQ                 ; CLEAR IRQ
          EI                         ; ENABLE INTERRUPTS
          CALL   TXT_LOAD            ; LOAD TEXT INTO BUFFER
          CALL   LCD_RES             ; CLEAR THE LCD
          POP    IMR                 ; RESTORE IMR
          RET                        ; RETURN FROM INTERRUPT
;
;-------------------------------------------------------------------------;
;          TAKE THE ASCII TEXT AND LOAD IT INTO THE BUFFER                 ;
;-------------------------------------------------------------------------;
TXT_LOAD:
          CLR    R6                  ; RESET BYTE COUNTER
          LD     R7,#BUFF_BEG        ; POINT TO ASCII BUFFER
LOAD:
          TM     IRQ,#POLL_SIO       ; BYTE IN SIO?
          JR     Z,LOAD              ; LOAD SOME MORE
          LD     @R7,SIO             ; STORE AT THIS BUFFER LOCATION
          AND    IRQ,#IRQ3_RES       ; RESET IRQ3
          INC    R7                  ; INC BUFFER ADDRESS
          INC    R6                  ; INC BYTE COUNTER
          CP     R6,#16              ; SIXTEEN BYTES YET?
          JR     LT,LOAD             ;  NO, KEEP GOING
          RET                        ; RETURN TO CALLER
```

```
;-------------------------------------------------------------------------;
;       TAKE THE ASCII FROM THE BUFFER AND LOAD THE LCD                   ;
;-------------------------------------------------------------------------;
LCD_LOAD:
        CALL    LCD_OUT             ; OUTPUT CONTENTS OF BUFFER TO LCD
        AND     PORT_0,#RS_LOW      ; TAKE RS LOW
        LD      PORT_2,#DD_RAM_2    ; LOAD STARTING ADDRESS FOR LINE 2
        OR      PORT_0,#E_HIGH      ; TAKE ENABLE HIGH
        AND     PORT_0,#E_LOW       ; TAKE ENABLE LOW
        CALL    LCD_OUT             ; OUTPUT TO LCD
        RET
LCD_OUT:
        OR      PORT_0,#RS_HIGH     ; TAKE RS HIGH
        CLR     R6                  ; RESET BYTE COUNTER
LCD_LOOP:
        LD      PORT_2,@R7          ; FETCH ASCII FROM BUFFER
        OR      PORT_0,#E_HIGH      ; TAKE ENABLE HIGH
        AND     PORT_0,#E_LOW       ; TAKE ENABLE LOW
        CALL    WAIT_2              ; WAIT AWHILE
        INC     R6                  ; INC BYTE COUNTER
        INC     R7                  ; INC BUFFER ADDRESS
        CP      R6,#8               ; EIGHT BYTES YET?
        JR      LT,LCD_LOOP         ;  NO, GO AGAIN
        RET                         ; RETURN
;-------------------------------------------------------------------------;
;       DELAY LOOP FOR LCD WRITES                                         ;
;-------------------------------------------------------------------------;
WAIT_2:
        LD      R4,#CONST_1         ; THIS DELAY NOT
BUSY:
        DJNZ    R5,BUSY             ; QUITE AS LONG
        RET                         ; RETURN TO CALLER
;
;
;
;
;
;-------------------------------------------------------------------------;
;       DELAY LOOP FOR LCD INITIALIZATION                                 ;
;-------------------------------------------------------------------------;
WAIT_1:
        PUSH    RP                  ; SAVE RP
        SRP     #MYREG_0            ; POINT TO 00H
        LD      R4,#CONST_1         ; LOAD FOR DELAY
LCDELY_1:
        LD      R5,#CONST_2         ; LCD'S ARE SLOW!
LCDELY_2:
        DJNZ    R5,LCDELY_2         ; DONE YET?
        DJNZ    R4,LCDELY_1         ;
        POP     RP                  ; RESTORE RP
        RET                         ; RETURN TO CALLER
;
;-------------------------------------------------------------------------;
;       INTERNAL LCD MESSAGES                                             ;
;-------------------------------------------------------------------------;
LCD_MSG:
        .BLOCK 16
        .ASCII 'ZILOG Z8s ARE OK'
        .END
```

THE VERSATILE Z86C08: THREE KEY FEATURES OF THIS Z8® MICROCONTROLLER

IF YOU NEED D/A CONVERSION, OR A ZERO CROSSING DETECTOR, OR A CURRENT SENSING DEVICE ... USE THE Z86C08'S DUAL COMPARATOR.

DUAL ANALOG COMPARATOR

Using the dual analog comparators on the Z86C08 in conjunction with several on-chip features, provides a cost effective way to monitor power failures and frequency excursions (comparator used as a zero crossing detector), as a blood pressure tester and digital readout (comparator used as a A/D converter), or as a current sensing device in automotive design to detect and subsequently shut off any short circuiting of relays, lights, monitors, etc.

In many microcontroller applications, the digital designer is often concerned with sampling and controlling non-digital elements within the system. However, when the designer is forced to deviate from the precise world of TTL logic and regulated 5 volt supplies, frequently, microcontroller architectures and specifications fall short in the areas of cost sensitivity and consumer orientation. Therefore, using the analog comparators in these specific areas are a few of the reliable, inexpensive design applications for the Z86C08.

Comparator Basics

The dual comparators share a common inverting terminal with non-inverting terminals bonded directly to external I/O ports (Figure 1). The comparators are enabled by a bit in the I/O port mode/control register. If bit D1 of R247 is zero, then the comparators are in digital mode. If D1 is one, then they are in analog mode. With the comparators disabled, the I/O ports are available for normal activities. These particular I/O ports can be used to generate external interrupt requests to the Z8®. With the comparators enabled, interrupts can also be generated.

The ideal comparator is a three terminal device (Figure 2). V1 is a non-inverting terminal. Signals entering at V2, the inverting terminal, exit V_{OUT} 180° out of phase. Since a comparator is essentially an operational amplifier, it has an associated gain. The open look gain (no feedback) of a comparator is defined as the Voltage Out (V_{OUT}) over the Differential Input Voltage. The Differential Input Voltage is the voltage at the non-inverting input with respect to the inverting input. Thus gain is:

GAIN = V_{OUT}/V1 – (V2) = Voltage Out/Differential Input Volt

The Inset Offset Voltage, the difference between V1 and V2, forces V_{OUT} to a specified level. The Input Offset Voltage is typically below 50 mV.

Zero Crossing Detect Applications

The dual comparator can be used as a zero crossing detector to monitor 110 VAC (or other power line parameters) and its frequency (Figure 3). Each time the voltage passes through zero an interrupt is generated. The outputs of the comparators on the Z86C08 connect directly to the on-chip CPU. When using the comparators to detect zero crossing of the signal, interrupts are generated at every crossing of a signal, interrupts are generated at every crossing. Interrupt subroutines can then calculate period and phase angle relationships between any two analog signals. The phase angle being critical when calculating power factor in power line circuits.

In the case of 110 VAC, 60 Hz power line, an interrupt is generated every 1/120 of a second. This means that whenever the monitor stops (no interrupts), there is a power fail or other problem which can be translated by a control device recovery action (Figure 4).

Frequency checks can also be made by zero crossing detection. Whenever frequency drifts from the normal monitoring zero points, interrupts are either increased (higher frequency) or decreased (lower frequency) from the norm. If necessary, appropriate action is then taken.

Zero Crossing Detect Applications (Cont'd.)

Another application is threshold detection for low voltage battery operated devices. Whenever the VBB drops below the Zener reference voltage level, an interrupt is generated to alert a control device or alarm.

The addition of two on-chip counter/timers further complement the above mentioned applications. Crystal precision timing is done on the period of zero crossings. The sum or difference of two separate analog signals can then be calculated. For example, negative or positive feedback is returned from the Z86C08 in closed loop calculations. In power circuits, a time-of-day clock could be implemented with a timer. Then, date and time of power failures and frequency excursions can be recorded. CMOS technology allows for battery backup.

Analog to Digital (A/D) Conversion

Accurate low speed A/D conversion is implemented with the Z86C08 using the dual slope or ratiometric method. With this method, a dv/dt is applied to the inverting terminal of a comparator. The analog input (V_{INPUT}) signal is applied to the non-inverting terminal. The charge rate of the RC circuit is a dv/dt (Figure 5). As V_{REF} ramps upward from zero volts during time T1, V_{REF} will exceed V_{INPUT}. This causes the comparator to change state and produce an interrupt. By using the on-chip timer, time T1 can be quickly determined.

The RC circuit is immediately discharged over fixed time T2 (Figure 6), where T2 is determined by the time constant Tc = RCn. Since the product of RC is only an approximate indicator of discharge time, a value of n should be multiplied to improve accuracy. A general guideline should equate n to 1.4. Then, T2 = 1.4 RC. The dual slope A/D converter measures voltage by converting voltage into time intervals. Or,

$T2/T1 = V_{INPUT}/V_{REF}$, then, $V_{INPUT} = V_{REF} = T2/T1$

By using an I/O port on the Z86C08 as the V_{REF} input, interrupts generated by the comparators can alternately switch V_{REF} ON or OFF to perform the conversions.

Example: Blood Pressure Tester

A pressure transducer in a blood pressure tester is a good example of the dual slope A/D conversion method. A minimum system consists of display logic, Z86C08 circuitry and a transducer signal input (Figure 7). P00 outputs the appropriate signal to the RC ramp circuit of the V_{REF} input. The output from the pressure transducer (Figure 8) is a linear voltage response to the applied pressure. This signal is input to An2, the non-inverting terminal of the comparator.

In this configuration, the sampling cycle for the A/D conversion begins when a logical 1 is output on P00 and a timer is enabled. When the comparator transitions, an interrupt is generated, the timer is stopped and P00 is toggled to discharge the RC circuit. By storing the count T1 and resetting the timer, the converter is now ready to take another sample. The value of V_{IN} is mathematically determined later and software algorithms are used to determine corresponding pressure.

The display is driven from a simple multiplexer circuit. The Z86C08 can sink large I_{OC} currents which reduces or eliminates buffering.

Current Sensing

The dual comparator is used as a current sensing device in many application areas, e.g., in automotive relays, lights, monitors, etc. In the automotive arena, current sensing is used in a typical case as shown in Figure 9a. If the functional block shorts, then current (I) surges causing voltage (V) to fall. When V reaches 2.5V, the comparator triggers an interrupt which allows software to enable an emergency shut off.

Comparator Basics

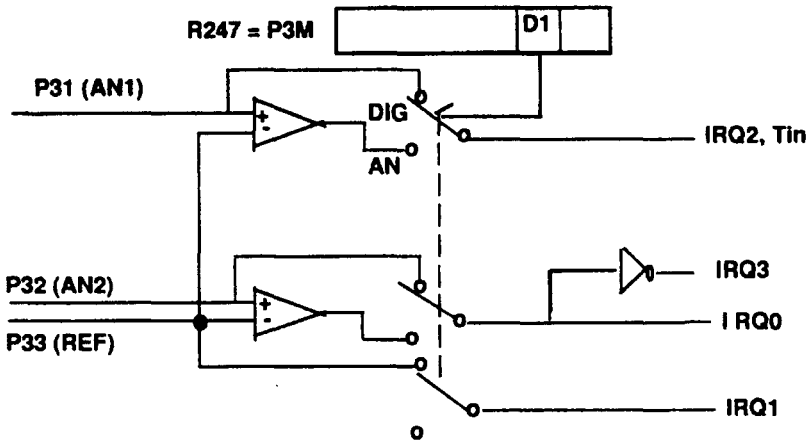


Figure 1. Dual Analog Comparator

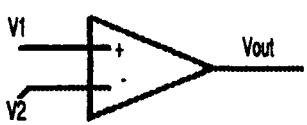


Figure 2. Ideal Comparator

Zero Crossing Detect Applications

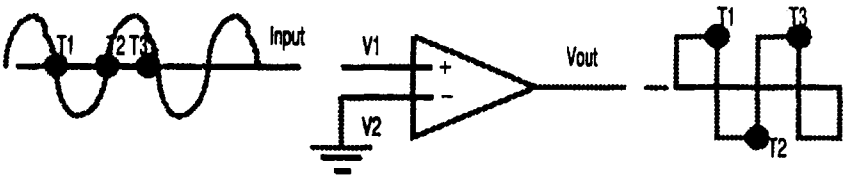


Figure 3. Zero Crossing Detector

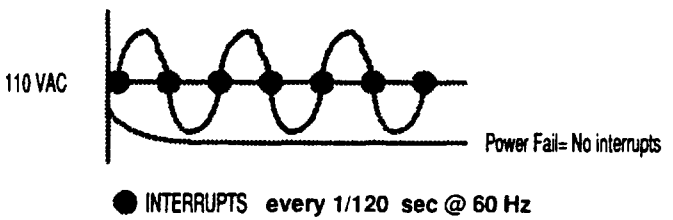


Figure 4. Interrupt After Power Failure

Analog to Digital (A/D) Conversion

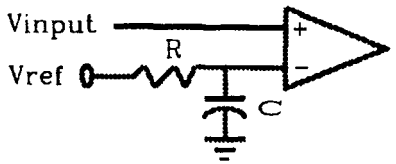


Figure 5. A/D Converter

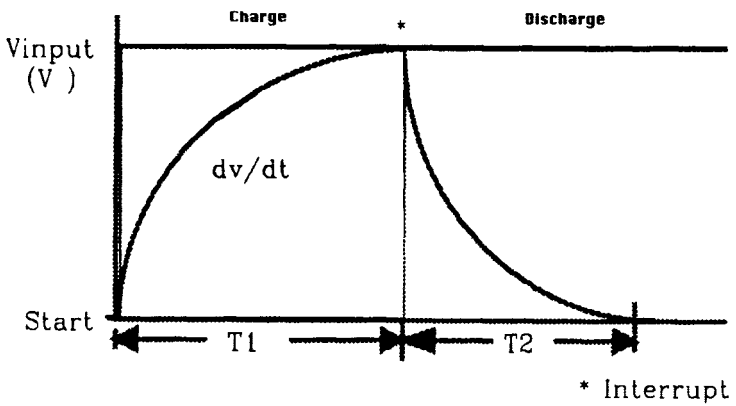


Figure 6. Voltage vs. Time

Blood Pressure Tester

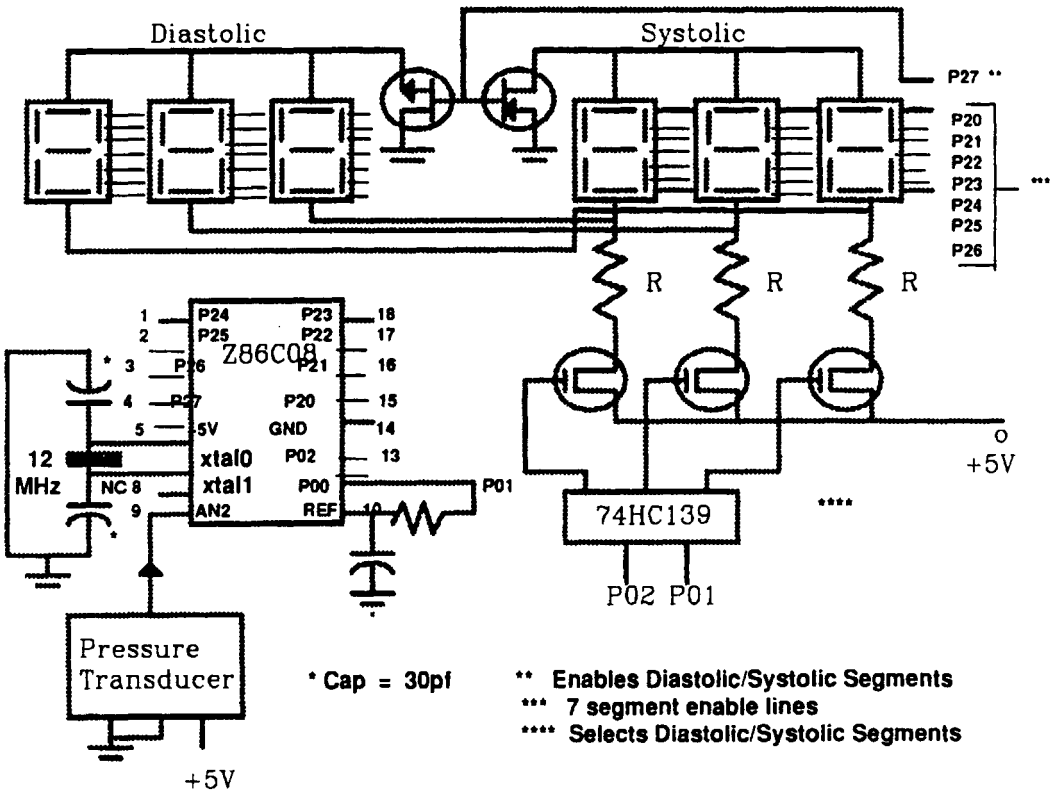


Figure 7. A/D Blood Pressure Test and Readout

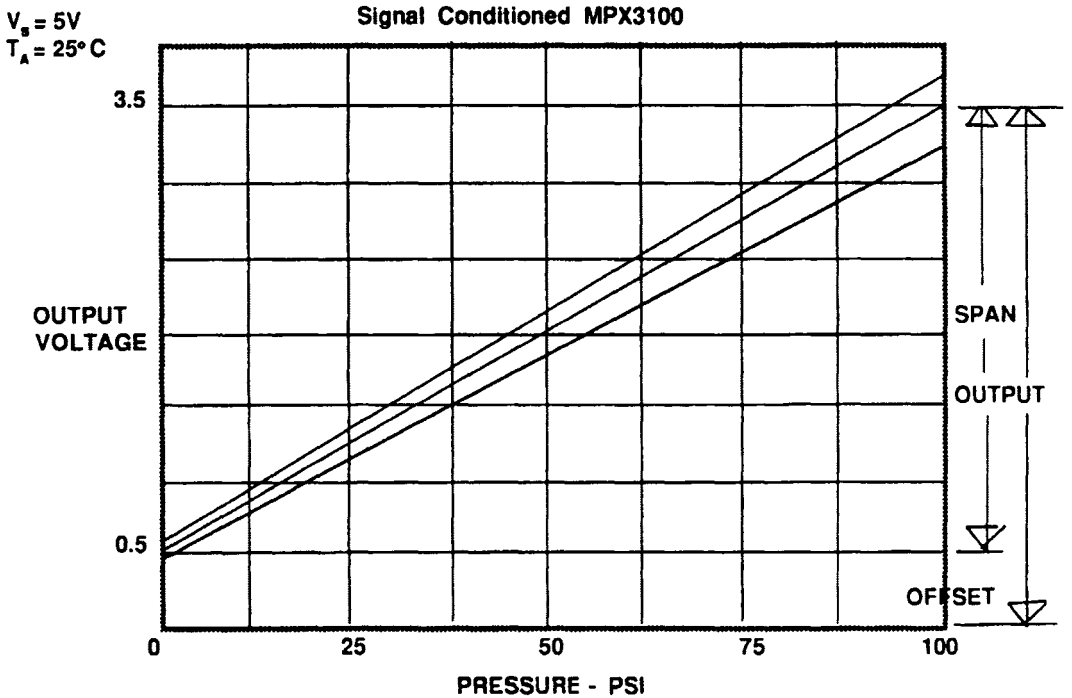


Figure 8. Silicon Pressure Transducer

Current Sensing

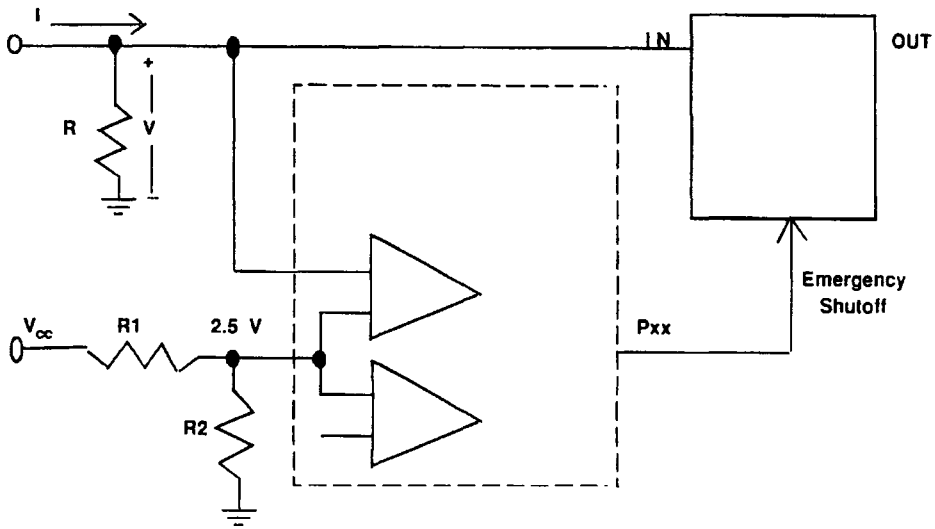


Figure 9a. Current Sensing

Note:

R is large compared to the equivalent impedance of the Functional Block input. R1 and R2 are user-selectable and are generally in a 10K to 100K range of power dissipation considerations. R1 and R2 are determined from the following formula (Figure 9b.).

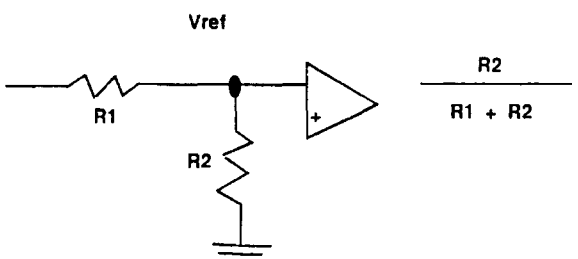


Figure 9b. Power Dissipation Formula

DESIGNING A LOW-COST THERMAL PRINTER

USING THE Z86E21 TO CONTROL THE OPERATING CURRENT ON LOW-COST THERMAL PRINTERS PROVIDES DESIGN FLEXIBILITY AND HELPS SAFEGUARD PERFORMANCE.

INTRODUCTION

Compact and low-cost thermal printers are popular for many applications: point of sale equipment; medical and industrial instrumentation; micro-fax and micro-printers for the consumer markets, as well as other applications. Each application is unique with different operating environments and performance requirements. In this application example, the Z86E21 is used as a common controller for thermal printers. The hardware and software flexibility and performance potential of the Z86E21 make it ideal for this type of application. This Application Note provides design engineers with information on how the Z86E21 interfaces with thermal printers, creating a platform from which they can customize their designs.

Mechanical Considerations

Thermal printers have several common, yet important attributes. Most important is a thermal print head which has a number of resistive heating elements. When heated, these elements produce an image on heat sensitive paper. Secondly, thermal printers must advance the print head across the paper media. It must make provisions to advance the paper after the head has traversed the width of the paper. Depending upon the cost and complexity of the thermal printer used, the mechanism for paper advance can be an independent motor and gear drive assembly, or simply a ratchet driven line feed for every print head carriage return.

Additionally, there are mechanical features which enable interconnect to the controller circuit. Also, there are provisions for securing the assembly to a case and paper cutter. The printer must supply sufficient pressure on the paper, sandwiched between the platen and roller, as well as maintain pressure when the paper is torn across a fixed, serrated cutter bar. Figure 1 shows a typical thermal printer. Note the threaded, head feed screw attached via a gear system to stepper motor #1. Motor #2 advances the paper as required by rotating the platen through a gear drive.

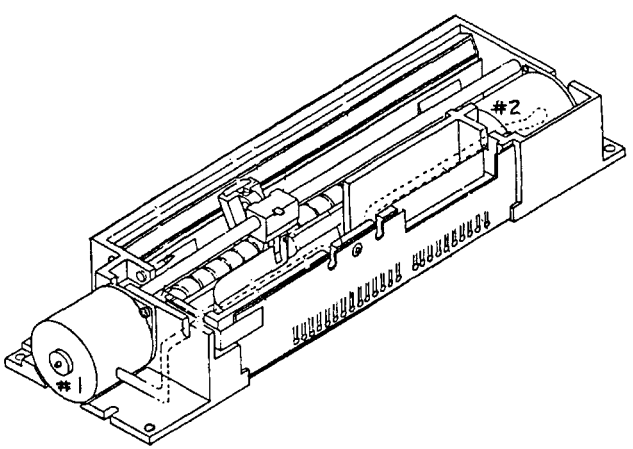


Figure 1. Thermal Printer

Electrical Considerations

The key elements to controlling thermal printers are in properly synchronizing the movement of the print head and paper to the required application. Also, as in any closed loop control system, the printer must deliver a status signal back to the controller. Since paper movement is continuous, it has no electronic analog of "begin and end", except for "paper out" or "paper jammed" conditions. To close the loop, thermal printers define a position for the print head as "home", and give an active Low feedback status when "home." Normally, this is at the far left of the carriage span.

Four-phase stepper motors are commonly used to transport the print head. If the printer is to print bi-directionally, a second stepper motor is used for paper advance. The equivalent circuit for the four-phase stepper motor, and the required support components are shown in Figure 2. The four diodes are used to discharge the inductive kick which is generated when the transistors are switched off. The Zener diode provides threshold for this action. Table 1 shows typical phase drive sequences for clockwise and counter clockwise rotational motion of a four-phase stepper motor (note equivalent Hex code values).

Table 1. Typical Phase Drive Sequences

Clockwise			Counter Clockwise		
Step	Phases	Hex Code	Step	Phases	Hex Code
1	1101	C	1	0011	3
2	0110	6	2	0110	6
3	0011	3	3	1100	C
4	1001	9	4	1001	9

Thermal print heads are arranged in a typical column of seven, eight, or nine elements. Each element is called a dot. Figure 3 shows the arrangement of an 8-dot head.

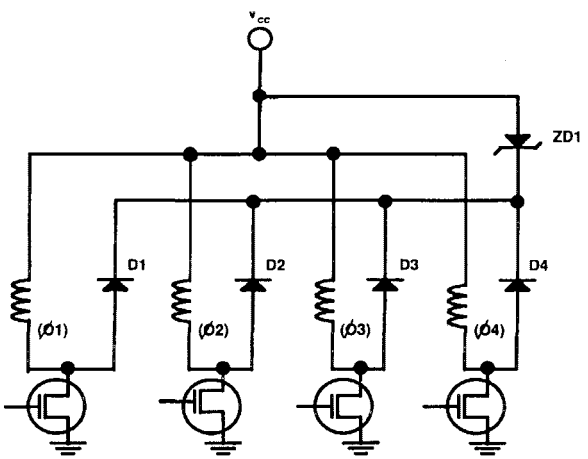


Figure 2. Equivalent Circuit for Four-Phase Stepper Motor

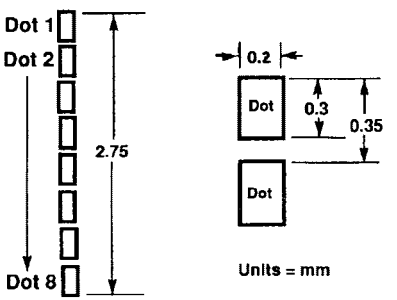


Figure 3. 8-Dot Head Arrangement

Since the print head traverses the width of the paper on the head feed screw, the electronic signals required to operate the head are delivered via a flexible printed circuit (typically copper traces sandwiched between mylar). Each dot has a closely controlled value of resistance. This resistance ranges between 14 and 18 Ohms in compact thermal printers. Heat is generated by a given dot through I^2R dissipation.

A dot has a maximum applied energy specification and a recommended or rated specification. Printer manufacturers describe this energy in joules, or more closely, milli-joules (mj). One joule per second is one watt. Energy is usually expressed with a time base. Understanding this concept is essential in thermal printer design. The print circuit must not exceed the energy rating of a dot. Simultaneously, it must control the duration (dissipation) to ensure proper print quality and protection of the print head elements (dots).

For example, a dot has a resistance of 16 ohms, a rated energy spec of 2.10 mj, and a maximum energy spec of 2.52 mj, at 5 volts. The pulse width (time) required to apply the rated energy in milliseconds is given by the equation in Figure 4.

Figure 4. Thermal Energy Equation

$$t = R * E / V2$$

Where: R = Head Resistance (Ohms)
E = Applied Energy (mj)
V = Voltage (Volts)
t = Pulse Width (ms)

The Thermal Energy Equation yields a required pulse width of 1.34 ms to apply the rated energy of the head. The value of E, the applied energy in the equation, needs to reflect the effects that ambient temperature fluctuations induce. The equation to compensate the applied energy used in deriving pulse width is given in Figure 5.

Figure 5. Ambient Temp Compensation Equation

$$E = E_0 (1 + T_0 - T/100)$$

Where: E = Applied Energy (mj)
E_0 = Rated Energy (mj)
T_0 = 24 °C
T = Ambient Temperature (°C)

As illustrated above in the Figure 5 equation, as ambient temperature decreases, supplied energy must be increased. Thus, the pulse width is increased to maintain print quality.

Printing a Character

Again, the key element to controlling a thermal printer is synchronizing the head movement to the print application. Figure 6 is a timing chart that shows the combination of the head feed motor steps with the applied energy pulses in printing the character "2."

Using Figure 6 and starting from the left (Step 1), no print elements are activated. In Step 2, the print head advances from its previous position and dots two and eight are activated. This "step then print . . . step then print . . . etc." process is repeated until the character is complete. If each step took 4 ms, this character would be complete in five steps, or 20 ms.

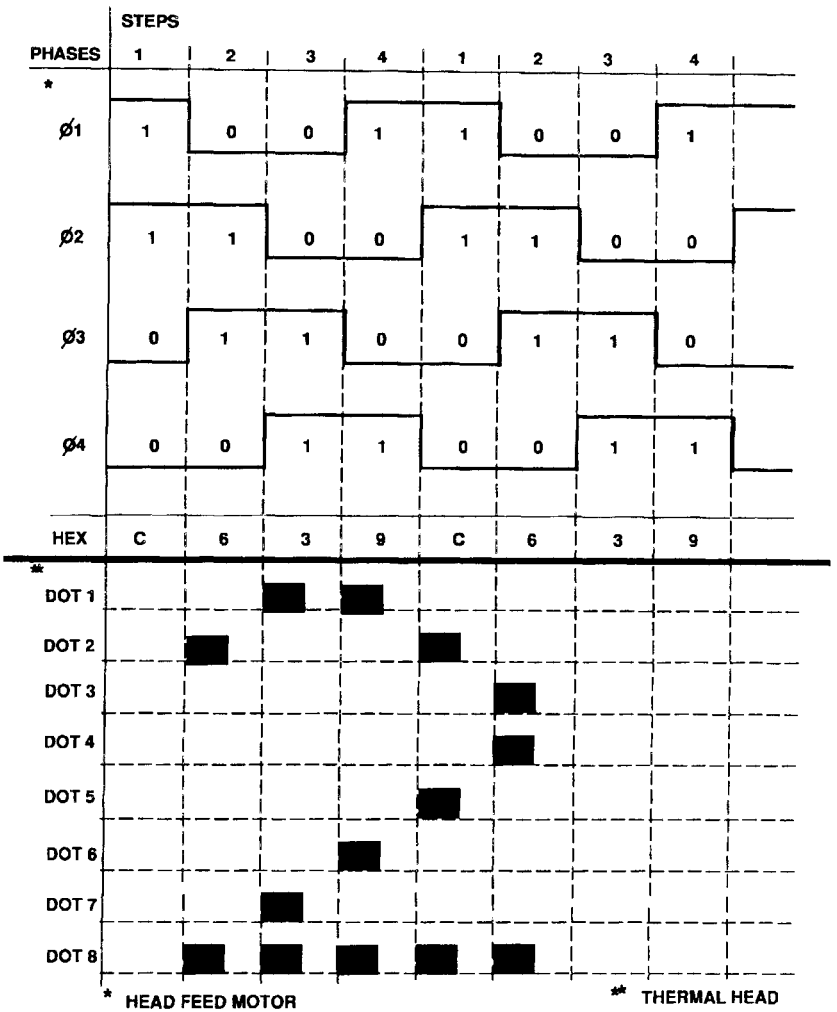

STEPS
PHASES
1 2 3 4 1 2 3 4
*
Ø1 1 0 0 1 1 0 0 1
Ø2 1 1 0 0 1 1 0 0
Ø3 0 1 1 0 0 1 1 0
Ø4 0 0 1 1 0 0 1 1
HEX C 6 3 9 C 6 3 9
**
DOT 1
DOT 2
DOT 3
DOT 4
DOT 5
DOT 6
DOT 7
DOT 8
* HEAD FEED MOTOR
** THERMAL HEAD

Figure 6. Printing the Character "2"

Z86E21 Controller Application

Ports 0 and 1 are used on the Z86E21. Low cost 18-pin, bipolar, octal peripheral array drivers (ULN2801A) are used to interface TTL logic signals from the Z86E21 to the printer (Figure 7). The outputs from the ULN2801A's are capable of sinking the 500 mA currents that the stepper motor windings source. Additionally, an active dot can typically source 300 to 400 mA of current. The equivalent circuit of the ULN2801A is shown in Figure 8. Note the inclusion of discharge diodes and the use of Zener diodes as discussed in the Electrical Considerations section (see Figure 2).

When the print head reaches the far left position of the carriage travel, it is in the home position. The home pin is actively Low when this status is present (Figure 7). The 0.1 µf capacitor (C1) debounces the event. The 2.2K resistor provides an RC time constant and current limit. The 1K resistor is a pull-up for the Z86E21 input. In this example, a home status produces a falling edge interrupt. The 5K port pull-ups provide sufficient bias current to the bipolar inverters of the ULN2801A's.

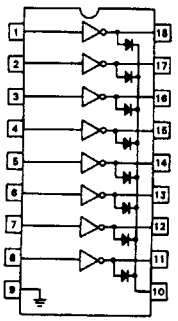


Figure 8. ULN2801A Equivalent Circuit

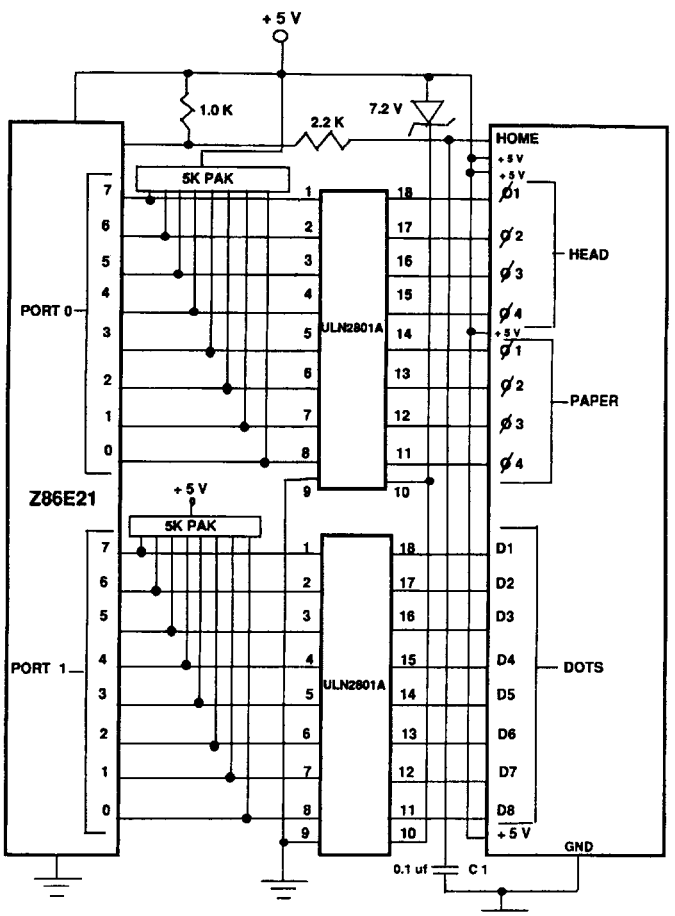


Figure 7. Z86E21 to 8-Dot Stepper Motor Control Interface

Print Head Protection

Depending upon the application environment and operating conditions, the designer must consider the degree of print head protection required. Bad resets, power fails, ESD jolts, and printing errors can potentially produce conditions whereby the head is left on too long. Depending on the maximum energy rating, the heads can be permanently damaged after only seconds of carrying current continuously. The designer should consider these various, potentially damaging, scenarios and plan to avoid or effectively manage them. One helpful design method is to install a MOSFET in the ULN2801A to V_{CC} path. An external watch-dog circuit could time the power-on duration in the print elements. In environments where ESD is a concern, the designer may consider using the Z86E40, a Z8 Family controller comparable to the Z86E21 but with the added feature of an on-chip watch-dog timer circuit. Multiple approaches to print head protection are worth the cost and effort.

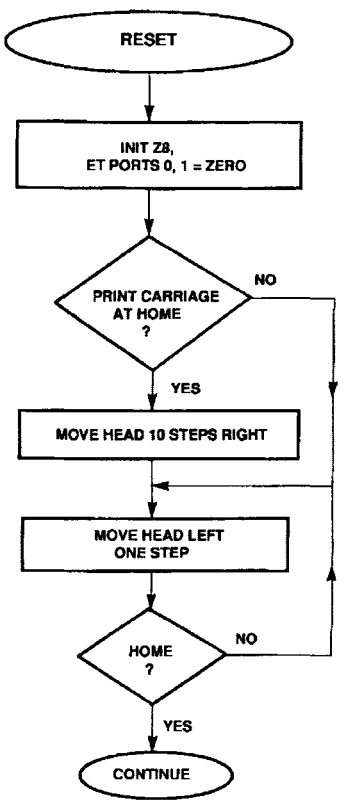


Figure 9a. Initialized Coding Flowchart

Initializing and Print Coding Flowcharts

The coding necessary to make this interface operate is shown in the Figure 9a flowchart. First, the Z86E21 is initialized and Ports 0 and 1 are set to zero. Counter/Timer 1 of the Z8 is used to provide a main task interrupt clock of 100 µs. This task clock yields the millisecond order signals for incrementing stepper motors and activating thermal printer heads with the proper accuracy. Continuing the flow, the print head carriage is "homed." To do this, the home status is first polled for a Low. Then, actions are taken to move the head in order to generate a falling edge interrupt for IRQ2, Pin 32.

Once home status is secure, Figure 9b flowchart shows typical print coding flow. A character is available once it has reached an internal FIFO stack in the Z8 register file. This FIFO stack is maintained by the interrupts which handle the received characters on the serial or parallel interface. The ASCII character is then converted to an index address in the ROM where the bit map for that particular character is located. This bit map is then printed one column at a time, sequentially incrementing the head position one step.

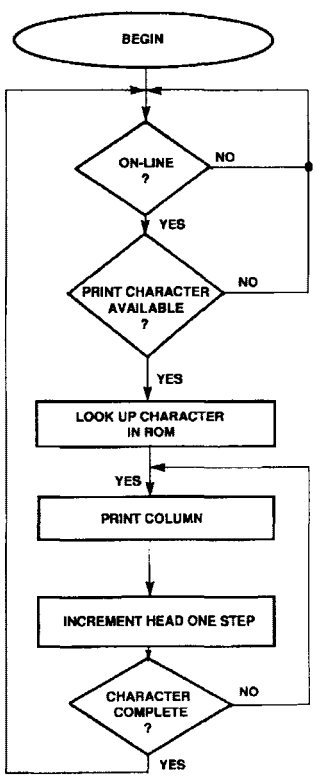


Figure 9b. Print Coding Flowchart

Z86E21 Flexibility

The Z86E21 further complements this application when considering the host interface possibilities available to the designer. The Z8 has UART on-board to easily implement an RS232-style serial interface to a host. Note that the Z8 is only required to receive data, thus RxD is the only active UART pin. Flow control comes from P31, where an active High condition flags printer not ready (Figure 10).

Another possible parallel interface is the Centronics. With 32 potential I/O signals on the Z8, 17 of which are used for the printer interface, only a subset of the Centronics interface can be adapted directly to the Z8. The designer could adapt full interfaces if required by the application, through additional logic (Table 2).

The Z8 has two-wire handshake logic built-in, greatly simplifying data flow. The asterisks (*) in Table 2 show the boundary between a simple parallel interface, and a complex one.

Table 2. The Parallel Interface

#	Name	Transmitter	Function
1	/STB	Host	Active Low (/) indicates valid data
2	D0	Host	Parallel data byte
3	D1	Host	
4	D2	Host	
5	D3	Host	
6	D4	Host	
7	D5	Host	
8	D6	Host	
9	D7	Host	
10	/ACK	Printer	Active Low (/) indicates data received
11	BSY	Printer	off line, error, or handling data
12	PE	Printer	Printer out of paper
13	SLCT	Printer	Active high, printer is on, and selected
14	/ALF	Host	Auto Line Feed
15	/INT	Host	Reset
16	/ERR	Printer	General Machine Error
17	/SLN	Host	Select-in, from from host

Z86E21 Flexibility (Continued)

Figure 10 illustrates the parallel interface. Data lines are pulled up to improve drive characteristics on the bus. The /STB signal easily maps into the Z8 /DAV2 function, where a falling edge generates a service interrupt for the current valid data byte. The BSY strobe output from the Z8 closely matches the RDY2 function. Refer to the Z8 input handshake functions in the Z86E21 Product Specification for further details. The /ACK strobe can be implemented via software in the character interrupt subroutine. Figure 11 shows the ideal handshake timing implemented by the software.

Conclusion

With a better understanding of how the Z86E21 interfaces with thermal printers, designers can take advantage of the flexibility and performance potential of this low cost Z8 microcontroller.

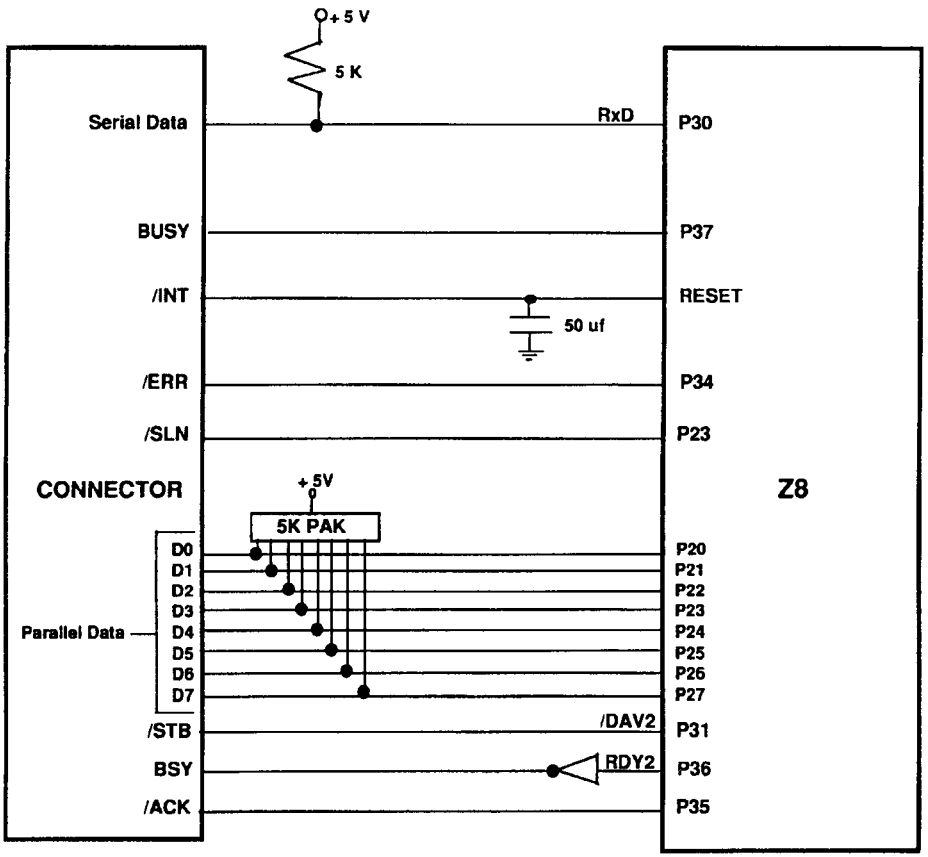
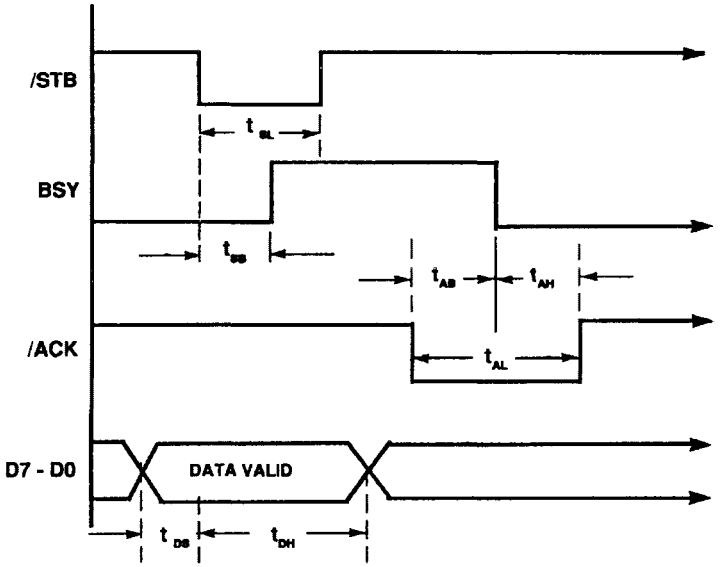


Figure 10. Z8 Implementation of a RS232 Interface



t_{SL} = 0.5 us min t_{DH} = 0.5 us min t_{AH} = 6 us typ
t_{SB} = 2.8 us typ t_{AL} = 10 us typ
t_{DS} = 0 us min t_{AB} = 4 us typ

Figure 11. Parallel Handshake Timing

ZILOG DOMESTIC SALES OFFICES AND TECHNICAL CENTERS

CALIFORNIA
Agoura 818-707-2160
Campbell 408-370-8120
Irvine 714-453-9701
San Diego 619-658-0391

COLORADO
Boulder 303-938-9906

FLORIDA
Clearwater 813-725-8400

GEORGIA
Duluth 770-931-4022

ILLINOIS
Schaumburg 847-517-8080

MASSACHUSETTS
North Chelmsford 508-251-0300

MINNESOTA
Minneapolis 612-944-0737

OHIO
Independence 216-447-1480

OREGON
Portland 503-274-6250

PENNSYLVANIA
Horsham 215-784-0805

TEXAS
Dallas 214-987-9987

INTERNATIONAL SALES OFFICES

CANADA
Toronto 905-850-2377

CHINA
Shenzhen 86-755-2220869
..... 86-755-2220873
Shanghai 86-21-6415-8158
..... Rm. 5204

GERMANY
Munich 49-8967-2045
Sömmerda 49-3634-38389

JAPAN
Tokyo 81-3-5272-0230

HONG KONG
Kowloon 85-2-2723-8979

KOREA
Seoul 82-2-577-3272

SINGAPORE
Singapore 65-2357155

TAIWAN
Taipei 886-2-741-3125

UNITED KINGDOM
Maidenhead 44-1628-392-00

© 1997 by Zilog, Inc. All rights reserved. No part of this document may be copied or reproduced in any form or by any means without the prior written consent of Zilog, Inc. The information in this document is subject to change without notice. Devices sold by Zilog, Inc. are covered by warranty and patent indemnification provisions appearing in Zilog, Inc. Terms and Conditions of Sale only. Zilog, Inc. makes no warranty, express, statutory, implied or by description, regarding the information set forth herein or regarding the freedom of the described devices from intellectual property infringement. Zilog, Inc. makes no warranty of merchantability or fitness for any purpose. Zilog, Inc. shall not be responsible for any errors that may appear in this document. Zilog, Inc. makes no commitment to update or keep current the information contained in this document.

Zilog's products are not authorized for use as critical components in life support devices or systems unless a specific written agreement pertaining to such intended use is executed between the customer and Zilog prior to use. Life support devices or systems are those which are intended for surgical implantation into the body, or which sustains life whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in significant injury to the user.

Zilog, Inc., 210 East Hacienda Ave.
Campbell, CA 95008-6600
Telephone (408) 370-8000
FAX (408) 370-8056
Internet: http://www.zilog.com

SALES REPRESENTATIVES AND DISTRIBUTORS

U.S., CANADIAN & PUERTO RICAN REPRESENTATIVES

ALABAMA

Huntsville
Alabama Bits, Inc ..... (205) 534-4020

ARIZONA

Scottsdale
Thom Luke Sales, Inc. ..... (602) 451-5400

CALIFORNIA

Calabasas
Infinity Sales ..... (818) 880-6480

Irvine
Infinity Sales ..... (714) 833-0300

San Diego
Addem ..... (619) 794-1999

Santa Clara
Phase II Technical Sales ..... (408) 980-0414

COLORADO

Englewood
Thorson Rocky Mountain ..... (303) 773-6300

CONNECTICUT

Wallingford
Advanced Technical Sales ..... (203) 284-9762

FLORIDA

Altamonte Springs
Semtronic Associates, Inc. ..... (407) 831-8233

Clearwater
Semtronic Associates, Inc. ..... (813) 461-4675

Fort Lauderdale
Semtronic Associates, Inc. ..... (305) 731-2484

GEORGIA

Norcross
BITS ..... (770) 564-5599

IDAHO

Eagle
Thorson Rocky Mountain ..... (208) 939-4345

ILLINOIS

Hoffman Estates
Victory Sales, Inc. ..... (847) 490-0300

IOWA

Cedar Rapids
Advanced Technical Sales ..... (319) 393-8280

KANSAS

Olathe
Advanced Technical Sales ..... (913) 782-8702

MARYLAND

Annapolis
Electronic Engineering & Sales ..... (410) 255-9686

MASSACHUSETTS

North Reading
Advanced Technical Sales ..... (508) 664-0888

MICHIGAN

Novi
Rathsburg Associates, Inc. ..... (810) 615-4000

MINNESOTA

Minneapolis
Professional Sales for Industry ..... (612) 944-8545

MISSOURI

Bridgeton
Advanced Technical Sales ..... (314) 291-5003

NORTH CAROLINA

Charlotte
BITS ..... (704) 336-7887

Raleigh
BITS ..... (919) 676-1880

NEW JERSEY

Cherry Hill
Tritek ..... (609) 667-0200

OHIO

Independence
Rathsburg Associates, Inc. ..... (216) 447-8825

OKLAHOMA

Broken Arrow
Nova Marketing, Inc. ..... (918) 660-5105

SALES REPRESENTATIVES AND DISTRIBUTORS

OREGON

Portland
Phase II Technical Sales ..... (503) 643-6455

TEXAS

Austin
Nova Marketing, Inc. ..... (512) 343-2321

Dallas
Nova Marketing, Inc. ..... (214) 265-4600

Houston
Nova Marketing, Inc. ..... (281) 240-8082

UTAH

Salt Lake City
Thorson Rocky Mountain ..... (801) 264-9665

WASHINGTON

Kirkland
Phase II Technical Sales ..... (206) 821-8313

WISCONSIN

Brookfield
Victory Sales, Inc. ..... (414) 789-5770

CANADA

British Columbia
J-Squared Technologies, Inc. ..... (604) 473-4666

Ontario
J-Squared Technologies, Inc. ..... (905) 672-2030

Ottawa
J-Squared Technologies, Inc. ..... (613) 592-9540

Quebec
J-Squared Technologies, Inc. ..... (514) 747-1211

PUERTO RICO

Rio Piedras
Semtronic Associates, Inc. ..... (809) 766-0700

U.S. AND CANADIAN DISTRIBUTORS

NATIONWIDE

Anthem Electronics ..... (408) 453-1200
Arrow Electronics ..... (516) 391-1300
Newark Electronics 1-800-367-3573
Zeus Electronics 1-800-524-4735

SALES REPRESENTATIVES AND DISTRIBUTORS

CENTRAL AND SOUTH AMERICA REPRESENTATIVES

MEXICO

Guadalajara
Nova Marketing 011-523-121-9772

BRAZIL

Sao Paulo
Nishicom 011-55-11-524-3533
Graftec 011-55-11-826-1666

ASIA-PACIFIC REPRESENTATIVES

AUSTRALIA
GEC Electronics Division 61-2-638-1888

CHINA

Beijing
China Electronics Appliance Corp. 755-335-4214

HONG KONG
Electrocon Products, Ltd. 852-2481-6022
Components Agent, Ltd. 852-2487-8826
MEMEC, (Asia Pacific) Ltd. 852-2410-2780
TLG Electronics, Ltd. 852-2388-7613

INDIA

Bangalore
Maxvale 91-80-286-6762

Bombay
Maxvale 91-22-850-9165

New Delhi
Maxvale (S) Pte. Ltd. 91-11-622-5122

INDONESIA

Jakarta
Orchard Technology 62-21-798-2762

JAPAN

Tokyo
International Semiconductor, Inc. 81-3-3264-3301
Kaga Electronics Co., Ltd. 81-3-3942-6244
Kanematsu Elec. Components Corp... 81-3-3779-7811
Teksel Co., Ltd. 81-3-5467-9000

KOREA
ENC-Korea 822-523-2220
MEMEC, (Asia Pacific) Ltd. 822-786-8180

MALAYSIA
Kuala Lumpor 60-3-703-8498

Penang
L.T. Electronics Ltd. 60-4-656-2895

NEW ZEALAND
GEC Electronics Division 64-9-526-0107

PHILIPPINES
Alexan Commercial 632-241-9493

SINGAPORE
Electronics Resources Limited 65-298-0888
Future Electronics 65-479-1300
Orchard Technology (S) Pte. Ltd. 65-299-7811

TAIWAN (ROC)
Acer Sertek, Inc. 886-2-501-0055
ASEC Int'l. Corporation 886-2-786-6677
MEMEC, (Asia Pacific) Ltd. 886-2-760-2028
Promate Electronics Co. Ltd. 886-2-659-0303

THAILAND
ER (Thailand) Co., Ltd 66-2-370-1280

SALES REPRESENTATIVES AND DISTRIBUTORS

EUROPE

AUSTRIA

Vienna
Avnet/Electronic 2000 43-222-9113853
EBV Elektronik GMBH 43-1-8941774

CZECH REPUBLIC

Ceske Mezirizi
ECOM spol 42-443-92202

GERMANY

Bad Camberg
Thesys/AE 49-6434-5041
Berlin
Avnet/Electronic 2000 49-30-2148820
EBV Elektronik GMBH 49-30-3421041
Burgwedel
EBV Elektronik GMBH 49-5139-80870
Dortmund
Future GMBH 49-231-9750480
Duesseldorf
Thesys/AE 49-211-5360214
Eppstein/Taunus
EBV Elektronik GMBH 49-6198-59200
Erfurt
Future GMBH 49-361-420870
Thesys 49-361-4278101
Erkrath
Avnet/Electronic 2000 49-211-920030
Frankfurt
Avnet/Electronic 2000 49-69-9738040
Gerlingen
Avnet/Electronic 2000 49-7156-3560
Hamburg
Avnet/Electronic 2000 49-40-6969520
Idstein
Future GMBH 49-6126-93210
Kaarst
EBV Elektronik GMBH 49-2131-96770
Langenhagen
Future GMBH 49-511-725620
Leonberg
EBV Elektronik GMBH 49-7152-30090
Muenchen
Avnet/Electronic 2000 49-89-4511001
EBV Elektronik GMBH 49-89-991140
Future GMBH 49-89-957270
Thesys A/E 49-89-99355866
Nuernberg
Avnet/Electronic 2000 49-911-931490
Quickborn
Future GMBH 49-4106-71021
Stuttgart
Future GMBH 49-711-830380
Thesys/AE 49-711-9889100
Weissbach
EBV Elektronik GMBH 49-36426-22486

ISRAEL

Petach Tekva
Migvan Technologies & Eng. Ltd. 972-3-9240-784

ITALY

Ancona
Avnet EMG S.R.L. 39-71-7819699
Cinisello
EBV Elektronik 39-2-66017111
Firenze
Avnet EMG S.R.L. 39-55-4360392
Milano
Avnet EMG S.R.L. 39-2-381901
EBV Elektronik S.R.L. 39-2-660961
Silver Star Ltd. 39-2-661251
Modena
Avnet EMG S.R.L. 39-59-342322
EBV Elektronik S.R.L. 39-59-356080
Napoli
Avnet EMG S.R.L. 39-81-5591477
Roma
Avnet EMG S.R.L. 39-6-4131151
EBV Elektronik S.R.L. 39-6-2253367
Torino
Avnet EMG S.R.L. 39-11-3112348
EBV Elektronik S.R.L. 39-11-2161531
Treviso
Avnet EMG S.R.L. 39-423-722675

POLAND

Warsaw
Gamma Ltd. 48-2-6639887

PORTUGAL

Amadora
Amitron-Arrow 35-1-4714806

SALES REPRESENTATIVES AND DISTRIBUTORS

RUSSIA

Vyborg
Gamma Ltd. 781-278-31509

St. Petersburg
Gamma Ltd. 812-552-62-75

SPAIN

Barcelona
Amitron-Arrow S.A. 34-3-4907494

Madrid
Amitron-Arrow S.A. 34-1-3043040

SWITZERLAND

Dietikon
EBV Elektronik GMBH 41-1-7456161

Lausanne
EBV Elektronik AG 41-21-3112804

Regensdorf
Eurodis AG 41-1-8413111

TURKEY

Istanbul
Elektro A.S. 90-216-4181270

SECTION AND TITLE	PART NUMBER
CORPORATE	
Master Selection Guide	*MS96COR3200*
Superintegration Products Guide	*PG96COR2101*
Quality and Reliability Report	*QR96COR2004*
DSP	
Digital Signal Processor Databook	*DB95DSP0105*
Z89C00 DSP User's Manual & Algorithms	*DC-8294-02*
DSP Mag Strip Flyer	*FL95DSP0100*
DSP Single-Chip Caller ID Flyer	*FL95DSP0200*
DSP Application Notes Handbook	*DB96DSP0300*
New DSP Family Flyer	*FL96DSP0300*
Small/Home Office Flyer	*FL96DSP0300*
Wavetable Synth Flyer	*FL96DSP0400*
KEYBOARD/MOUSE	
Keyboard Databook	*DC-8304-01*
Keyboard Flyer	*FLY5K001003*
Keyboard Module Flyer	*FY96KEY0400*
LOW VOLTAGE	
IR Remote Controllers Databook	*DB97LV00100*
IR User Guide Jungle Book	*DC-8324-01*
IR Remote Controllers Flyer	*FL95LV00102*
Zilog Database of IR Codes Flyer	*DC-5631-00*
MASS STORAGE	
Mass Storage Databook	*DC-8303-01*
Z86C95 DSP User Manual	*DC-8595-02*
PCMCIA	
PCMCIA Z86017/Z16017 Reference Manual	*UM95HDD0101*
SCC HIGH-SPEED	
SCC High-Speed Databook	*DC-8314-01*
SCC High-Speed Z16C30/Z16C32 User's Manual	*DC-8350-01*
Datacom Migration Brochure	*BR95SCC0102*
ZNW2000 PC WAN Developer's Kit Brochure	*DC-5613-00*
ZNW2000 PC WAN Developer's Kit U.M.	*UM95Z800101*

SECTION AND TITLE	PART NUMBER
SERIAL COMMUNICATION CONTROLLERS	
SCC Databook with Application Notes	*DC-8316-00*
SCC™/ESCC™ & ISCC™ User's Manual	*UM95SCC0102*
Zilog's Turnkey PCMCIA-SCSI Flyer	*FL495PC0100*
GeoPort Flyer	*FLY95SCC101*
SCSCI Databook	*DB95SCC0100*
TV	
Digital Television Controllers Databook	*DC-8308-01*
Z86129/30/31 Product Specification	*DS96TEL0201*
Z86129 V-Chip Flyer	*FL96TEL0700*
Digital TV Controller User's Manual	*DC-8284-01*
Internet Brochrue	*BR96TEL0100*
Z89300 Brochure	*DC-5608-01*
Z90100 Multi Language OSD Flyer	*FL95TEL0200*
Z90200 Multi Language OSD Flyer	*FL95TEL0400*
TV/Video Brochure	*DC-5567-01*
V-Chip Flyer	*FL96TEL0500*
Home Theater Flyer	*FL96TEL0600*
CABLE/SATELLITE	
Set-Top Box Solutions Flyer	*FL95CAB0100*
VOICE PROCESSORS	
Telephone Answering Device Databook	*DC-8300-03*
Voice Z89175/C67/176 Flyer	*FL95TAD0104*
WIRELESS	
Z2000 User Manual	*DC-8336-01*
10/25 Channel Cordless Phone Flyer	*FL96WRL0100*
ZPhone: The 900 MHz Cordless Phone Flyer	*FL96WRL0101*
Wireless Databook	*DB96WRL0101*

SECTION AND TITLE	PART NUMBER
Z8®	
8-Bit Embedded Controllers CD ROM	CD96Z80100
8-Bit Embedded Controllers Databook	DB96Z8X0200
Z8 Application Note Handbook	DB97Z8X0101
Z8 Discrete User's Manual	UM95Z800103
Z86LXX 2-Volt Flyer	FL96Z8X0800
Chip Carrier Brochure	DC-5672-00
Z8 CCP Emulator Brochure	DC-5759-00
Microcontroller Catalog	BR95DZ80102
Z8 Fast Design Flyer	FL96DZ80500
Z86C02/E02/L02 Flyer	FL96DZ80600
Z86E30/E31/E40 Flyer	FL96DZ80700
Z8 Microcontrollers Databook	DC-8305-03
32K ROM Z8 Brochure	DC-5601-01
BATTERY	
Battery Charging System Flyer	FL95BAT0100
Z80®	
Z80 Databook	DB95Z800101
Z80 User's Manual	DC-8309-01
Z180™	
Z180 Databook	DB971800100
Z180 User Manual	DC-8276-04
Z185/Z195 Product Specification	DS961850300
Z185 Flyer	FLY951800100
Z8S180 Emulator Flyer	FL951800200
Z182 Brochure	DC-5525-02
Z380™	
Z380 Product Specification	PS953800104
Z380 User's Manual	UM953800101
Z380 Flyer	DC-5580-03
Z8000®	
Z8000 Family Databook/User's Manual	DC-8319-00

Retro Workshop 果粉工作室

果粉藏書

ORDERING INFORMATION

Complete the following literature order form.

Mail To:

Zilog, Inc.
210 E. Hacienda Ave., M/S C1-0
Campbell, CA 95008-6600
Phone: (408) 370-8016
Fax: (408) 370-8056
Internet: http://www.zilog.com

SHIPMENT

Please allow four weeks for delivery.

PLEASE PRINT OR TYPE			
NAME			PHONE
COMPANY			
ADDRESS			
CITY	STATE	ZIP	COUNTRY

PART NUMBER	DOCUMENT TITLE	QTY.

DB97Z8X0101

Zilog, Inc.
210 East Hacienda Ave.
Campbell, CA 95008-6600
408-370-8000
Internet: http://www.zilog.com